Федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Курс лекций

дисциплины базовой части математического и естественнонаучного цикла Б.1

«Микропроцессорные системы»

(6 семестр)

Направление подготовки 09.03.01 Информатика и вычислительная техника

Профили: Автоматизированные системы обработки информации и управления, Вычислительные машины, комплексы, системы и сети

Авторский коллектив:

Доцент кафедры ВМСС, к.т.н. _____ С.Н. Михалин

«01» февраля 2017 г.

1. Основные термины

Микропроцессор (*CPU – central processor unit*) – программно-управляемое устройство, осуществляющее процесс обработки цифровой информации и управления им, построенное на одной или нескольких интегральных микросхемах.

Архитектура – общая конфигурация основных компонент микропроцессора, их главные возможности и характеристики, а также взаимосвязи.

Интерфейс – совокупность шин, сигналов, электронных схем и алгоритмов предназначенных для управления передачей информации между устройствами.

Системная шина – совокупность проводников, соединяющих микропроцессор с памятью и устройствами ввода-вывода. Обычно проводники шины объединяются в три группы:

- линии данных для передачи информации (**шина данных**);
- линии адреса, показывающие, откуда и куда передается информация (**адресная шина**);
- линии управления, регулирующие действия на шине (**шина управления**).

Микропроцессоры по признаку реализации архитектуры делят на:

1) **Принстонскую** (Фон Неймана, предложена в 1945 году), признаками архитектуры является одна шина доступа в ПЗУ и ОЗУ (одно адресное пространство), регистры имеют заранее определенное назначение;
2) **Гарвардскую** (Говард Эйкеном, конец 1930 года), признаками этой архитектуры является раздельные шины доступа к ОЗУ и ПЗУ, регистры однородны по назначению (универсальны, однотипны).

Примерно с середины 1980-х все микропроцессоры по внутренней архитектуре можно разделить на два типа:

1) ***RISC*** (*Reduced Instruction Set Computer*) – процессоры с сокращенной системой команд. Эти процессоры обычно имеют большой набор однородных регистров универсального назначения (РОНы), их система команд отличается простотой. В результате аппаратная реализация такой архитектуры позволяет выполнять инструкции, как правило, за один такт синхронизации, а длины самих команд унифицированы, кроме того, в арифметических операциях операндами могут быть только регистры.
2) ***CISC*** (*Complete Instruction Set Computer*) – процессоры с полным набором инструкций. Состав и назначение их регистров существенно неоднородны (количество регистров невелико). Команды создаются для удобства, а не эффективности исполнения (часто производят несколько простых действий). Это усложняет декодирование команд и как результат возрастает число тактов синхронизации необходимых для выполнения команды, длины команд различны.

Позже, примерно с 1997 года, по инициативе *Intel* появилась архитектура ***EPIC*** (*Explicitly Parallel Instruction Computing*) – «смесь» *RISC* и *CISC* архитектур (частые инструкции делаем короткими и выполняем за такт, «мощные» команды работают с отдельным массивом однородных регистров), основы этой идеи были реализованы в i80486 процессоре.

Микропроцессоры также можно классифицировать по назначению (универсальные и специальные), по разрядности (8-и, 16-и, 32-х, 64-х и 128-и), по производительности (число инструкций в единицу времени – *MIPS* /*million instruction per second*/), и т.д.

На сегодняшний день промышленность выпускает большое количество различных процессоров и их модификаций. Ведущими компаниями по производству микропроцессоров принято считать *Intel*, *AMD*, *Texas Instruments*, *Microchip*, *Atmel* и т.п.

Бурное развитие технологии приводит к тому, что нет смысла подробно изучать какие-либо процессоры (т.к. скорее всего за период обучения они морально устареют). Поэтому очевидно следует ограничиться только общими принципами построения, которые с развитием технологии меняются достаточно медленно и обычно носят цикличный характер. В табл. 1 представлено развитие семейства процессоров *Intel* 80x86.

Таблица 1. Развитие семейства процессоров i80x86

№	тип	разряд-ть *RG/D/A*	произв-ть, MIPS	технология, нм ($S_{крист}$, мм^2)	транзисторов, млн. (мощность, Вт)	Год выпуска
1	i8085	8/8/16	0.37	3 мкм	6.5e-3	1974
2	i8086, 4.77 МГц 8 МГц 10 МГц	16/16/20	0.33 0.66 0.75	3 мкм (30 или 16)	0.029 (1.75)	1978
3	i80286	16/16/24	2.6-5.5	1.5 мкм (49)	0.134	1982
4	i80386DX (*P*3)	32/32/32	8.9	1.5 мкм, 1.0 мкм (42)	0.275 (2 /33 МГц)	1985
5	i80486 (*P*4, 24) 486DX4	32/32/32	20–50 ~70	1.0, 0.8 мкм 600 (76)	1.19 1.6	1989 1994
6	*Pentium* (*P*5) (*P*54*C*), шина *PCI* (*P*55*C*), *MMX*	32/64/32	~110 ~200	800, 600 350 (90) 350 (141)	3.1 (16) 3.2 (10-15.5) 4.5 (17)	1993-94 1995 1997
7	*Pentium* II (*P*6)	32/64/36	~450	350 (204) 250 (131)	7.5 (43) 7.5 (27.1)	1997-98 1998
8	*Pentium* III (*P*6), *SSE*	32/64/32	2050	250, 180 (95) 130 (80)	9.5, 28 (38/1.1ГГц) 44 (32 /1.4 ГГц)	1999 2001
9	*P*4 (*P*68), PCI-E *-NetBurst, SSE*2 *-Prescott*, *SSE*3 *Pentium D*	32/64/32 32/64/36 64/64/36 64/64/36	~5000 9700 ~17100	180 (217) 130 (131) 90 (112) 65 (140)	42 (100 /2 ГГц) 55 (134 /3.4 ГГц) 125 (152 /3.8 ГГц) 376 (130 /3.6 ГГц)	2000 2002-07 2004 2005-06
10	*Core (Yonah) mobile* *Core* 2 *duo* | *quad* *Penryn , SSE*4.1	32/64/32 64/64/36 64/64/40	 11000/cor 10500/cor	65 (90.3) 65 (143 |286) 45 (107 |214)	*duo*: 151 (25/2ГГц) 291 |582 (130/3.3Г) 410 |820 (95 /3.0Г)	2006 2006-07 2007-08

Таблица 1. Развитие семейства процессоров i80x86 (продолжение)

№	Тип, (технорма), описание/особенности	Кэш /память	*TDP*, Вт ($S_{крист}$, мм^2) /транз., млн.	Графика	Год
11	*Nehalem Core i*3,*i*5,*i*7 *SSE*, *SSE*2,3, *SSSE*3, *SSE*4.1,4.2	L1: 32КБ x2/ядро L2: 256КБ /ядро		только *mobile*	
	*Core i*7 *Bloomfield* (45нм), 4 ядра, *LGA*1366, шина *QPI*	L3: 6 Мб /32 ГБ, 3xDDR3	130 (263) /731	нет	2008
	*Core i*7 *Lynnfield* (45нм), 4 ядра, *LGA*1156, *PCI-E*, шина *DMI*	L3: 8 Мб /16 ГБ, 2xDDR3	95 (296) /774		2009
12	*Westmere* (см. *Nehalem i*7)				
	*Core i*7 *Gulftown* (32нм), 6 ядер, *LGA*1366	L3: 12 Мб /32 ГБ, 3xDDR3	130 (248) /1168	нет	2010
	*Core i*7 *Arrandale* (32нм), 2 ядра *BGA*1288, *DMI*	L3: 4 Мб	35 (81+114) /382+177	45нм, 766 МГц	2010
13	*Sandy Bridge i*3,*i*5,*i*7 (32 нм) *SSE*1,2,3, *S*3,4.1,4.2, *AVX, AES* 4 ядра, *LGA*1155, *DMI* 2.0 2 ядра (*core i*3)	L1: 32КБ x2/ядро L2: 256КБ /ядро L3: 8 Мб L3: 3 Мб	 95 (216) /995 65(149) /624	 нет *HD*3000	2011
	*Sandy Bridge-E i*7, 6 ядер, *LGA*2011	L3: 15 Мб /64 ГБ, 4xDDR3	130 (435) /2270	нет	2011 -12
14	*Ivy Bridge i*7 (22 нм), 4 ядра *LGA*1155, *DMI* 2.0, *PCI-E* 3.0	L1: 32КБ x2/ядро L2: 256КБ /ядро	77 (160) /1400	*HD*4000 до 1.15 ГГц	2012

	*SSE*1,2,3, *S*3,4.1,4.2, *AVX*, *AES*	L3: 8 Мб			
	*Ivy Bridge-E i*7, 6 ядер *LGA*2011	L3: 12 Мб /64 ГБ, 4xDDR3	130 (256.5) /1860	нет	2013
15	*Haswell i*7 (22 нм), 4 ядра, *LGA*1150, *DMI* 2.0, *PCI-E* 3.0 *SSE*-4.2, *AVX*2, *AES*, *FMA*3, *BMI*1,2	L1: 32КБ x2/ядро L2: 256КБ /ядро L3: 8 Мб	84 (177) /1400	нет	2013
	Haswell-H, 4 ядра *BGA*-1364	L3: 6 Мб	65 (264+84) /1400	Iris Pro 5200 до 1.3 ГГц	2013
	Haswell-E, 8 ядер *LGA*2011-3	L3: 20 Мб /64 ГБ, 4xDDR4	140 (356) /2600	нет	2014
16	*Broadwell i*7 (14 нм), 4 ядра *LGA*2011, *BGA*-1364, *DMI* 2.0, *PCIE* 3.0 *SSE*-4.2, *AVX*3, *AES*, *FMA*3, *BMI*1,2	L1: 32КБ x2/ядро L2: 256КБ /ядро L3: 6 Мб eDRAM: 128 МБ /32 ГБ, 2xDDR3	65 (167) /1900	Iris Pro 6200 до 1.15 ГГц	2015
	Broadwell-E (14 нм), 6|8|10 ядер *LGA*2011-3	L3: 15|20|25 Мб /128 ГБ, 4xDDR4	140 (| |246) / | |3200	нет	2016
17	*Skylake-S i*7 (14 нм), 4 ядра *LGA*1151, *DMI* 3.0, *PCI-E* 3.0 *SSE*-4.2, *AVX*2, *AES*, *FMA*3, *BMI*1,2	L1: 32КБ x2/ядро L2: 256КБ /ядро L3: 8 Мб /64 ГБ, 2xDDR4	91 (123) /1350	*HD*530 до 1.15 ГГц	2015 -16
18	*Kaby Lake i*7 (14+ нм), 4 ядра *LGA*1151, *DMI* 3.0, *PCI-E* 3.0 *SSE*-4.2, *AVX*2, *AES*, *FMA*3, *BMI*1,2, *vPro*	L1: 32КБ x2/ядро L2: 256КБ /ядро L3: 8 Мб /64 ГБ, 2xDDR4	91 (126) /	*HD*630 до 1.15 ГГц	2017
19	*Coffee Lake i*7 (14++ нм), 6 ядер *LGA*1151, *DMI* 3.0, *PCI-E* 3.0 *SSE*-4.2, *AVX*2, *AVX*-512, *AES*, *FMA*3, *BMI*1,2, *vPro*	L1: 32КБ x2/ядро L2: 256КБ /ядро L3: 12 Мб /64 ГБ, 2xDDR4	95 (151) / 145 turbo	*UHD*630 до 1.2 ГГц	2018
20	*Cannon Lake* (10 нм) *Core i*3-8121*u*, 2 ядра *FCBGA*-1510	L1: 32КБ x2/ядро L2: 512КБ /ядро L3: 2 Мб /ядро /32 ГБ, 2xDDR4	15 (~70.5) /	*UHD*7xx	2018 -20
21	*Ice Lake*	L1i: 32КБ /ядро L1d: 48 КБ /ядро L2: 512КБ /ядро L3: 8 Мб		*UHD*8xx	202?

В современных микропроцессорных системах существует множество различных шин и интерфейсов, что обуславливается необходимостью связи с различными устройствами (быстрыми, медленными, с параллельным или последовательным обменом и т.п.). Шины "располагаются" по иерархическому принципу: каждая шина все больше отдаляется от процессора (становится более медленной) и каждая шина подключается к находящемуся выше ее уровню, объединяя различные компоненты системы (в частности – ПК):

1) Шина внутреннего кэша: самая быстрая шина, соединяет процессор и внутренний *L*1-кэш (понятие "кэш" будет разъяснено далее).
2) Системная шина: это шина второго уровня, соединяет подсистему памяти с чипсетом и процессором. В некоторых системах шины процессора и памяти представляют собой одно и то же. В ПК эта шина до 1998 г. работала с частотой до 66 МГц, а позже – 100, 133 и 200 МГц. В процессорах *Pentium* II реализована архитектура с двойной независимой шиной (*Dual Independent Bus*) – одна из них предназначена для доступа к основной памяти и называется передней шиной (*front side bus*), а вторая – для доступа к *L*2-кэшу и называется

задней шиной (*back side bus*). Наличие двух шин повышает производительность, т.к. процессор может одновременно получать данные с обеих шин. Отметим, что системную шину называют также основной шиной, шиной процессора и даже локальной шиной.

3) Локальная шина ввода-вывода: быстродействующая шина ввода-вывода используется для подключения быстрых периферийных устройств к памяти, чипсету и процессору. Такую шину используют видеокарты, дисковые накопители и сетевые интерфейсы. Типичный пример: *Peripheral Component Interconnect* (*PCI*), также *PCI Express*.
4) Универсальная последовательная шина (*Universal Serial Bus – USB*), позволяющая подключать до 127 медленных периферийных устройств с использованием хаба (*hub*). Пропускная способность версии 1.1 (1998 г): 12 Мбит/сек, версии 2.0 (2000 г): 480 Мбит/сек, версии 3.0 (2008 г): до 5 Гбит/сек. Чтобы гарантировать надёжную передачу данных интерфейс *USB* 3.0 использует избыточное кодирование (аналогично *Serial ATA*): один байт (8 бит) передаётся как 10 бит, что улучшает надёжность передачи в ущерб пропускной способности. Кроме того, надо помнить, что пропускная способность интерфейса делится между всеми подключенными к нему устройствами.
5) Скоростная последовательная шина *IEEE*-1394 (*FireWire*) предназначена для подключения потоковых устройств: цифровые камеры, принтеры, сканеры, телевизоры и т.д., требующих высокой пропускной способности (до 800 Мбит/сек).

Несколько шин ввода-вывода, соединяющие различные периферийные устройства с процессором, подключаются к системной шине с помощью моста (*bridge*), обычно реализованного в чипсете. Системный чипсет управляет всеми шинами и обеспечивает правильность взаимодействия устройств на системной шине (арбитраж). Каждая шина включает: шину данных и шину адреса. Конечно, имеются сигнальные линии для управления функционированием шины и сигнализации о доступности данных, которые можно объединить собирательным термином "шина управления". Деление шины на данных, адреса, управление условно, т.к. один и тот же физический проводник реализует несколько функций (в различные моменты времени и/или в разных режимах), но такое деление удобно для понимания и описания процессов работы микропроцессорных систем.

Системная шина реализована как набор проводников на материнской плате и должна соответствовать конкретному типу процессора. Именно процессор определяет характеристики системной шины. Вместе с тем, чем быстрее системная шина, тем быстрее должны быть остальные электронные компоненты системы. Технологические достижения позволяли повышать частоту процессора, а соответствие скорости системной шины требовало повышения быстродействия внешних компонентов, в основном – системной памяти, что было сопряжено со значительными трудностями и стоимостными ограничениями. Поэтому со времен процессора 80486DX2-50, у которого применялось удвоение частоты (МП работал с частотой 50 МГц, а системная шина на 25 МГц), для повышения производительности системы частоту МП получают умножением частоты шины на повышающий коэффициент (метод особенно хорошо работает благодаря наличию внутреннего *L*1-кэша, который удовлетворяет большинство обращений процессора к системной памяти).

На определенном этапе развития микропроцессоров понятие системной шины *FSB* потеряло первоначальный смысл (на примере семейства x86 это произошло с появлением *Pentium* II).

1.1 Обобщенная архитектура микропроцессорных систем

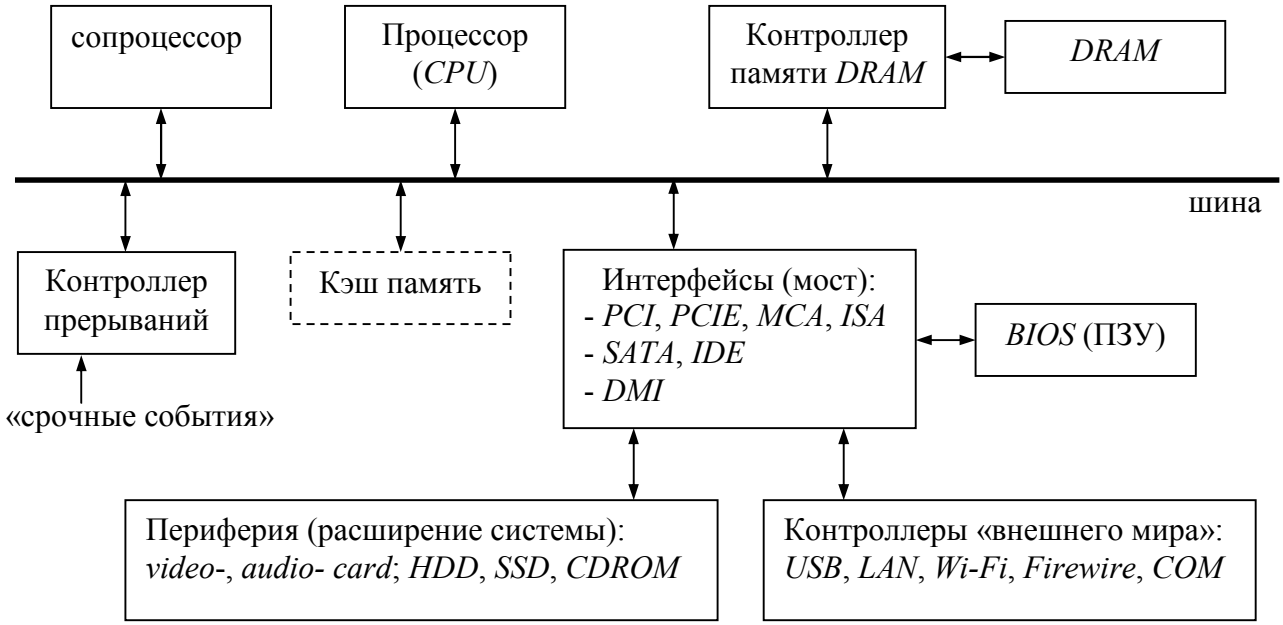


Рис. 1 – Обобщенная архитектура микропроцессорных систем

Шинная топология предполагает бесконечную пропускную способность шины (на практике в силу физических законов является «узким местом»). Поэтому с развитием ЭВМ архитектура систем и самих процессоров непрерывно эволюционирует. В частности, ресурсоемкие элементы интегрируют в состав процессора (есть возможность работать "в обход" системной шины, да и в интегральном исполнении такая шина обладает большей производительностью): сопроцессор, контроллер памяти, кэш-память, контроллер *PCI-E*, графический ускоритель и т.п.

Проблема ограниченной пропускной способности простейшей линии связи (дорожка на печатной плате, соединение на кристалле, отрезок кабеля и т.п.) связана с физическими свойствами материалов. Последние можно «трансформировать» в интегральные понятия индуктивность, емкость, сопротивление – рис.2. Дорожка как средство передачи сигнала представляет собой конструкцию двух проводников – металлическая полоска (фольга), диэлектрик (текстолит – основа платы или кремний и его оксид – основа кристалла) и обратный проводник (обычно лист фольги, покрывающий всю плату – *GND*). В первом приближении – имеем два электрода разделенных диэлектриком – получаем конденсатор, с другой стороны электроды с обоих концов замкнуты внутренним сопротивлением источника и сопротивлением нагрузки (приемник сигнала) – возникает контур – рамка и, следовательно, можно говорить об индуктивности. Фольга имеет конечную проводимость – имеется активное сопротивление (потери). Хотя более правильно, говорить о цепи с распределенными параметрами, т.к. длины волн распространяемого сигнала соизмеримы с длиной линии (мы всегда хотим достичь максимально возможной частоты тактирования, что соответствует уменьшению длины волны). Считается, что пределом проводной линии связи (витой пары) длиной 100 м является частота порядка 0.8 ГГц (полоса частот канала). Таким образом, в первом приближении канал связи можно рассматривать как ФНЧ. Например: для распространения в гипотетической линии связи с полосой частот 1 ГГц тактовых импульсов (меандр, преобразование Фурье есть бесконечное множество нечетных гармоник) требуется полоса "вмещающая" порядка 50 гармоник (для сохранения адекватного качества тактовых импульсов), т.е. тактовая частота, вероятно, не превосходит 20 МГц.

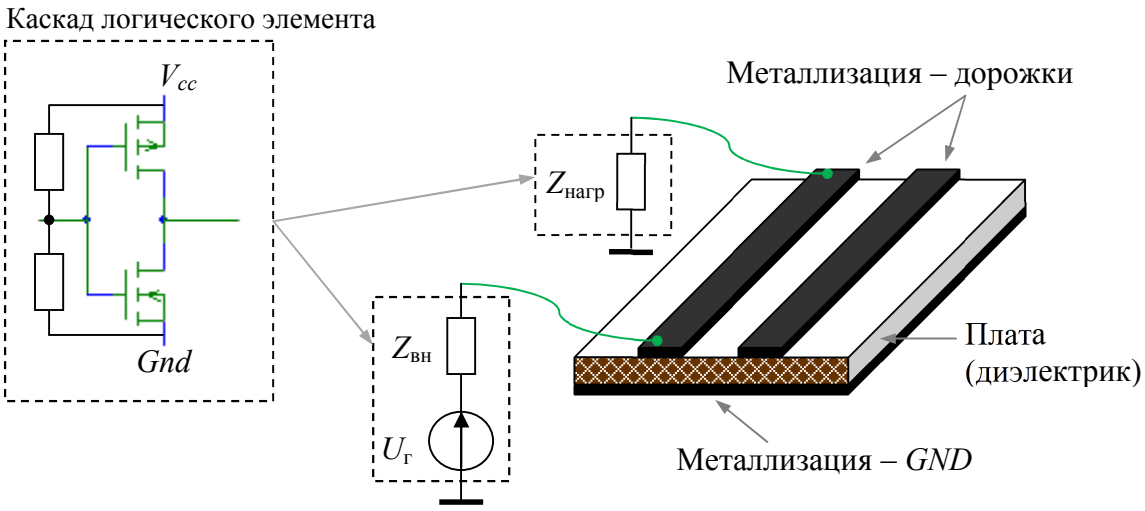


Рис. 2 – Представление линии связи на печатной плате

Следовательно, независимо от теории объясняющей физические процессы:

1) с точки зрения генератора: есть проблема формирования сигнала на реактивную нагрузку через канал со своими частотными характеристиками (проводя аналогию, вспомним прохождение сигнала прямоугольной формы через *RC* цепь (ФНЧ 1-го порядка) – фронт сигнала искажается, говорят «заваливается»);
2) с точки зрения приемника сигнала: образуемая системой проводников рамка является антенной, т.е. еще приемником электромагнитного излучения (например, вспомним закон Фарадея), при этом источниками электромагнитного излучения являются соседние дорожки (протекающий в проводнике ток создает магнитное поле), т.е. приемник и источник излучения находятся в непосредственной близости и их плоскости параллельны (это обеспечивает наилучшую паразитную связь).

Поэтому с ростом частоты протекающих токов и их величины растет эффективность излучения и соответственно прием электромагнитных сигналов и тем еще более широкополосной должна быть АЧХ системы передачи сигнала. Очевидно, емкость, индуктивность и активное сопротивление зависят от длины, ширины, толщины, удельной проводимости материала дорожки (обычно медь), диэлектрических свойств печатной платы (относительная диэлектрическая проницаемость, обычно в районе 4). Поэтому частоты тактирования шины на системной плате с типовыми размерами *ATX* стандарта не превышают 133-200 МГц, по аналогичным причинам ограничена скорость параллельного интерфейса *IDE* (который подразумевает использование плоских многожильных кабелей – шлейфов длиной несколько десятков сантиметров). Отметим, что у множества параллельно идущих пар проводников проблема носит кумулятивный характер. Это является одной из причин, что современные интерфейсы являются последовательными или представляют собой совокупность последовательных каналов обмена данными (нельзя также забывать об экономических аспектах). Наиболее яркими примерами служат: переход от *IDE* к *SATA*, от *FSB* к *QPI* и кольцевым шинам, от *LPT* к *COM* и *USB*.

Одной из действенных мер борьбы с проникновением в канал помеховых воздействий является применение дифференциальной схемы передачи сигнала – рис.3, что приводит к компенсации помеховых воздействий от соседних каналов и соответственно позволяет увеличить частоту переключений в канале без риска наведения помеховых ЭДС сравнимых с уровнем переключения логических элементов. Базовый протокол, который стал использоваться для последовательной передачи данных по дифференциальным парам, получил название *LVDS* (*Low Voltage Differential Signaling*). На основе протокола *LVDS* построены другие протоколы, имеющие сходные характеристики, среди которых можно отметить: *Infiniband*, *Ethernet*, *Hyper Transport*, *PCI Express*, *Fiberchannel*, *Firewire*, *Universal Serial Bus* (*USB*), *SSCSI* (*serial SCSI*), *SATA* (*serial ATA*).

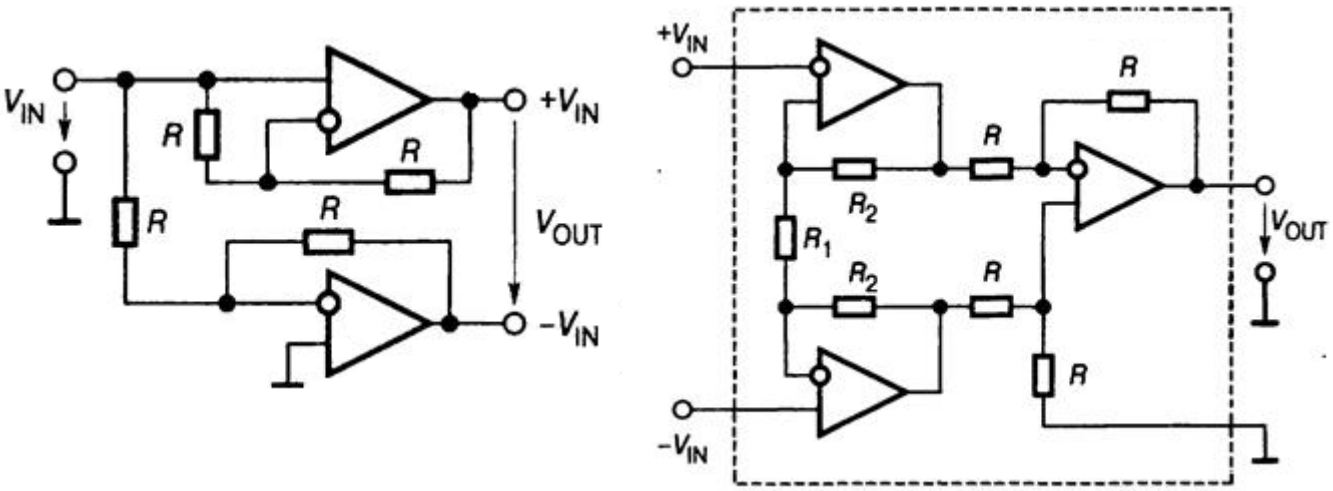


Рис. 3 – Драйверы дифференциального способа передачи сигнала (передатчик и приемник – слева и справа соответственно)

Эволюция производительности различных стандартизованных шин, применяемых в персональных ЭВМ (наиболее распространенных), представлена на рис.4[1].

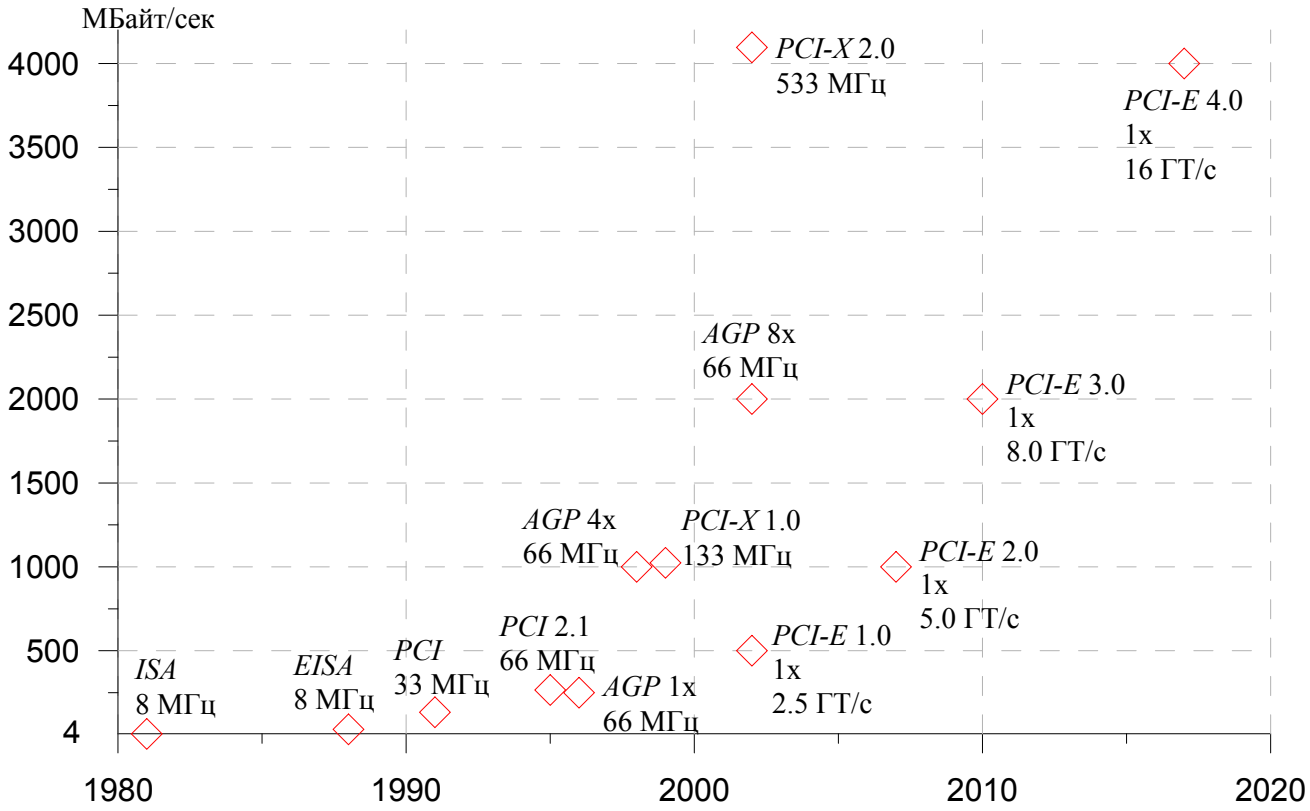


Рис. 4 – Производительность шин персональных ЭВМ

[1]Наиболее популярные шины, применяемые в ПЭВМ за последние полвека:
ISA (*Industry Standard Architecture*) – шина появилась на компьютерах в 1981 году. Последняя модификация в 1993 году добавила поддержку *Plug and Play*, которая позволяла ОС определять ресурсы, назначаемые для устройства.
PCI (*Peripheral Component Interconnect*) – появилась весной 1991 на смену шине *ISA* и ее аналогам, до 2010 года претерпела несколько модификаций (наиболее распространенная версия 2.1 – появилась в 1995 году).
PCI-E (*PCI Express*) – шина, также известная как 3*GIO* (*3rd Generation I/O*), появилась в июле 2002 года на смену шине *PCI*, на сегодняшний день – актуальная версия 3.0, в ближайшей перспективе – 4.0, в разработке 5.0. Шина допускает объединение (расширения полосы пропускания) до 32 каналов (32x).

Однако, даже обладая идеальными линиями связи (шинами, обеспечивающими быстрый обмен данными) микропроцессор не может рассчитывать на мгновенный отклик соединенных с ним устройств (в том числе и память). Дело в том, что элементарный транзистор на переключение тратит время и соответственно переключение их совокупности также требует времени – применительно к логическим вентилям это известно как задержка распространения сигнала, применительно к памяти – время доступа, латентность.

Для согласования времен доступа (скорости работы устройств на шине) и скорости работы микропроцессора необходимо вводить задержки (называются холостыми тактами – т.е. периоды бездействия процессора) либо применять буферизацию – рис.5. Суть буферизации состоит в накоплении определенного объема данных от медленного устройства с целью последующей обработки блочными методами с максимальной скоростью МП по сравнению со случаем без буфера – когда МП простаивает, ожидая данных от медленного устройства. Такой принцип буферизации называется буфером *FIFO* (*first in – first out*). На практике часто применяется двойная буферизация – два буфера одинакового размера – пока один заполняется, микропроцессор обрабатывает данные из другого, затем буферы «меняются местами» (процессор просто меняет адрес чтения данных).

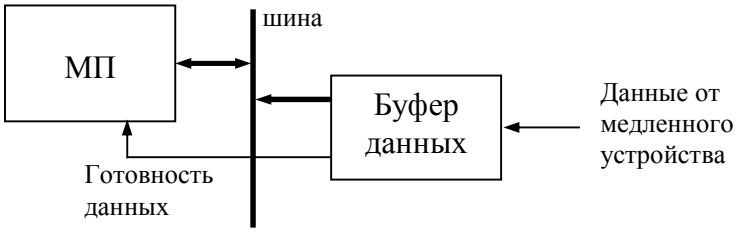


Рис. 5 – Буферизация потока данных (принцип *FIFO*)

Практические задачи часто требуют сохранения контекста (с целью временного хранения) – данных, которые потребуются позже. Причем таких операций сохранения может быть несколько (идущих друг за другом) и важен порядок их следования (т.е. в частности порядок обратной загрузки). В таких случаях эффективен другой тип буферизации – стек, реализующий принцип *LIFO* (*last in – first out*). Все процессоры имеют аппаратную реализацию стека, т.к. с его помощью обеспечивается эффективный вызов подпрограмм (в том числе рекурсивные вызовы) при ограниченном количестве регистров (внутренних ресурсов) процессора.

Частным случаем буферизации является кэш. С точки зрения микропроцессора обработка данных состоит в чтении исходных данных из оперативной памяти (ОЗУ), их модификации и записи обратно в ОЗУ (и необязательно поверх исходных данных). Пример: считали из памяти значение какой-то переменной, прибавили к нему какое-то число – записали обратно на то же место; считали массив, отсортировали по возрастанию – записали результат в память. Таким образом, оказывается, что программа часто работает не со всей ОЗУ целиком, а с относительно маленьким фрагментом (причем порядок чтения/записи может быть произвольным). Поскольку производительность ОЗУ всегда оказывается меньше способности (скорости) процессора обрабатывать данные (т.е. ОЗУ относительно процессора работает медленнее), то возникает идея сделать быструю память (время доступа на порядок меньше), которая дороже, сложнее в производстве, требует больше энергии и т.п., но небольшого объема (много меньшего чем объем ОЗУ). При обработке данных загружать их в эту быструю память. И далее процессор работает с ней, а не с относительно медленной ОЗУ (зато дешевой, больших объемов, меньше потребляющей и более простой в производстве). Процессор при необходимости чтения данных формирует соответствующий запрос на шину к памяти. Данные читаются из ОЗУ и попадают в быструю память и в процессор одновременно. Причем читается больше данных, чем процессор «просил», т.к. часто данные являются массивом, записью, т.е. располагаются в последовательных ячейках (соседних адресах). Таким образом, последующее обращение процессора по смежному адресу не вызовет обращения к медленному ОЗУ, а будет удовлетворено из быстрой памяти. Точно также поступают с записью результата – процессор

сохраняет результат в быструю память, из которой потом (или возможно одновременно) данные попадают в ОЗУ – существует несколько политик (способов) когерентности данных быстрой памяти и ОЗУ. Такая организация быстрой памяти называется кэшем. Конечно, существуют специальные алгоритмы, которые реализованы аппаратно («прозрачны» для микропроцессора и программиста) и решают задачи:

- соответствия данных кэша и ОЗУ;
- освобождения кэша (рано или поздно, следуя запросам процессора, малый объем кэша будет занят данными, а запросы к памяти от процессора будут продолжаться – требуется принять решение какие данные в кэше считать не нужными – будут удалены);
- быстрого ответа на вопрос: «есть ли запрашиваемые данные в кэше», если нет – необходимо обращение в ОЗУ.

Другим применением кэширования является необходимость предвыборки команд (поскольку последние тоже хранятся в относительно медленном ОЗУ). Процессор, работая быстрее памяти, требует постоянного поступления команд, а прямое чтение команд из ОЗУ приведет к ожиданию процессором их поступления. Поэтому команды читаются блоком в кэш, откуда процессор получает их по мере выполнения текущей программы. Новые команды загружаются в кэш блоками по мере его опустошения автоматически, не дожидаясь «требования» со стороны процессора (команды в памяти также хранятся друг за другом, исключение составляют переходы в программе и вызовы функций – тогда необходимо перезагружать кэш или предугадывать подобные ситуации для упреждения полной перезагрузки кэша команд, что является медленной и нежелательной операцией).

Специфика конструкции современных процессоров состоит в том, что система кэширования в *CPU* делается многоуровневой. Кэш первого уровня (самый «близкий» к ядру процессора) традиционно разделяется на две части (обычно половины): кэш инструкций (*L*1*I*) и кэш данных (*L*1*D*). Исторически (когда кэш был одноуровневым), это разделение предусматривается гарвардской архитектурой. В *L*1*I*, соответственно, аккумулируются только команды (с ним работает декодер команд процессора), а в *L*1*D* – только данные (они впоследствии, как правило, попадают во внутренние регистры процессора). Над *L*1 стоит кэш второго уровня – *L*2. Он, как правило, больше по объёму, и является уже «смешанным» – в нем располагаются и команды, и данные. Кэш третьего уровня (*L*3), полностью повторяет структуру *L*2 и его можно сделать довольно большим. *L*3 работает несколько медленнее *L*2, но всё равно быстрее чем ОЗУ. Тем не менее, алгоритм работы с многоуровневым кэшем в общих чертах не отличается от алгоритма работы с одноуровневым. Просто добавляются лишние итерации: сначала информация ищется в *L*1, если её там нет – в *L*2, потом – в *L*3, и уже потом, если ни на одном уровне кэша она не найдена – идёт обращение к основной памяти (ОЗУ).

2. Архитектура и программно-аппаратная модель процессора i8086

Для дальнейшего рассмотрения МП необходимо из множества их типов остановится на конкретной модели. Учитывая широкую распространенность *Intel* x86 совместимых процессоров, рассмотрим процессор i8086, отечественный аналог – микросхема К1810ВМ86.

*I*8086 представляет собой первый однокристальный 16-битовый МП, выполненный по *n*-МОП технологии 3 мкм в 1978 году. Кристалл микросхемы с геометрическими размерами 5.5x5.5 мм содержит около 29000 транзисторов и потребляет 1.75 Вт от однополярного источника питания +5 В (и это было «новшеством»).

На рис.6 представлена укрупненная структурная схема процессора и его условное графическое обозначение.

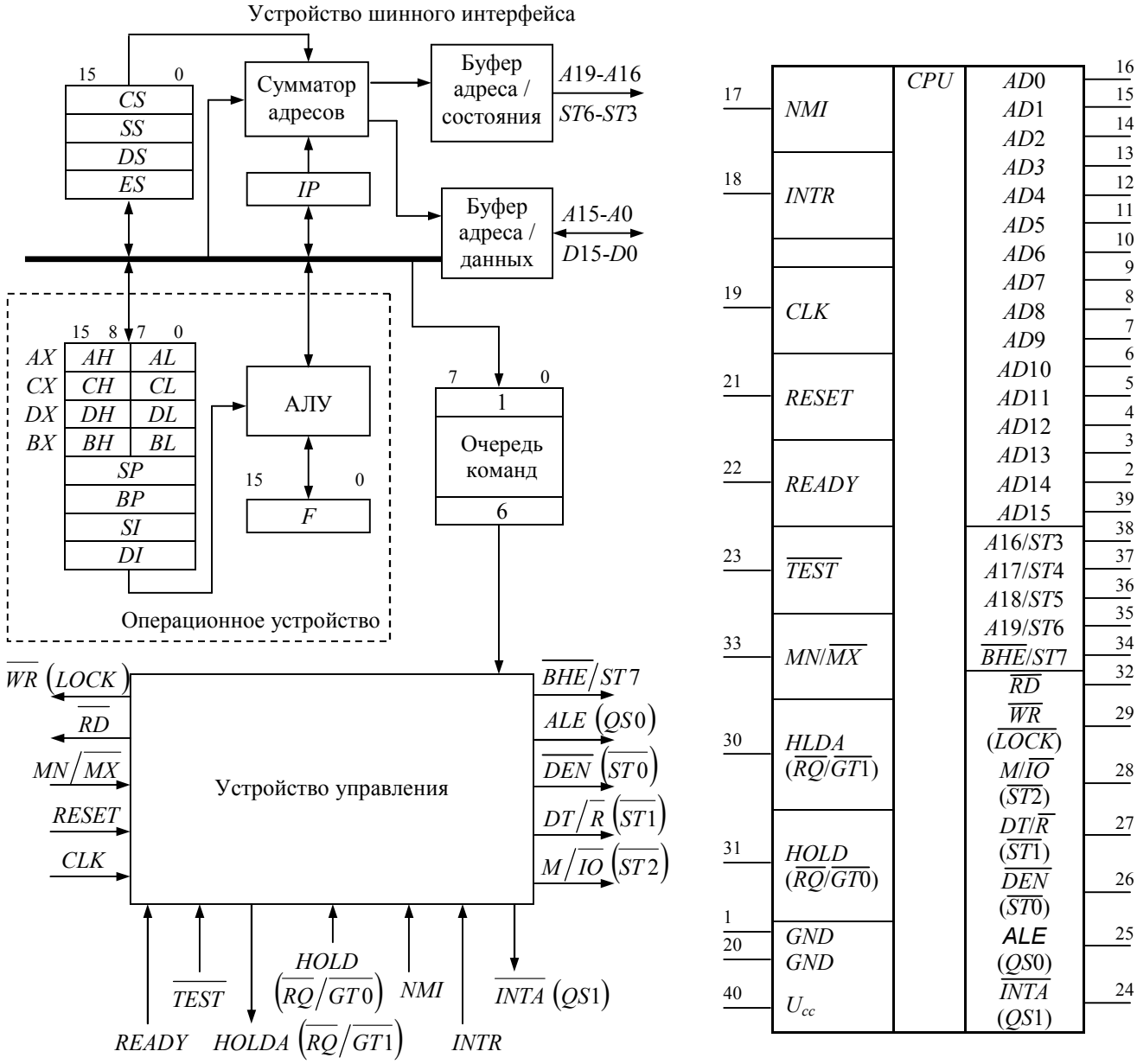


Рис. 6 – Структурная схема процессора *i*8086 и его условно-графическое обозначение

Назначение сигналов:

AD15-AD0 – мультиплексная (совмещенная) двунаправленная шина адреса/данных, по которой с разделением во времени передаются адресная информация и данные. В первом такте цикла шины – цикла обращения к ЗУ или внешнему устройству (ВУ) – МП выдает на эту шину младшие 16 бит адреса памяти или полный адрес внешнего устройства. Этот адрес обязательно должен быть зафиксирован и сохранен в течение всего цикла, для чего используется внешний регистр-защелка, куда записывается адресная информация с помощью строба адреса *ALE* (рис.7). Регистр-защелка должен иметь выходные буферы с тремя состояниями и обеспечивать малое время переключения при большой нагрузочной способности (например, этим требованиям отвечают 8-разрядные регистры ИР22, ИР23). Во второй половине цикла шины по линиям *AD*15-*AD*0 передаются адреса данных или байты команд, сопровождаемые стробом данных *DEN*.

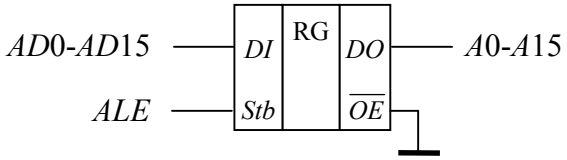


Рис. 7 – Применение регистра-защелки для демультиплексирования линий *AD*

A19/ST6-A16/ST3 – мультиплексные выходные линии адреса/состояния. В первом такте на эти линии выдаются старшие 4 бит адреса памяти, а при адресации ВУ – нули. В остальных тактах цикла шины МП выдает на эти линии сигналы состояния *ST*6-*ST*3. Для демультиплексирования адреса также необходимо применять регистр-защелку.

- Код на линиях *ST*4, *ST*3 определяет сегментный регистр, участвующий в формировании физического адреса памяти, т. е. указывает сегмент памяти, к которому производится обращение в текущем цикле (табл.2). Следует отметить, что при обращении к ВУ, когда сегментные регистры не участвуют в формировании адреса, устанавливается значение *ST*4=1, *ST*3=0.

Таблица 2

*ST*4 *ST*3	Сегментный регистр
0 0	*ES*
0 1	*SS*
1 0	*CS*
1 1	*DS*

Сигналы *ST*4, *ST*3 могут использоваться для расширения адресного пространства системы. В этом случае отдельный банк памяти объемом 1 Мбайт выделяется каждому из четырех сегментов. К линиям *ST*4, *ST*3 подключается дешифратор, который выбирает соответствующий банк памяти. Такой прием может обеспечить расширение адресной памяти до 4 Мбайт.

- Сигнал *ST*5 соответствует состоянию флага разрешения прерываний *IF*: 0 – прерывания запрещены, 1 – прерывания разрешены.
- Сигнал *ST*6 не используется и всегда равен нулю.

BHE – разрешение старшего байта. Формируется в первом такте цикла одновременно с адресной информацией. Активный сигнал нулевого уровня *BHE* означает, что по старшей половине шины адреса/данных (*AD*15-*AD*8) передаются 8-битовые данные. Сигнал *BHE* защелкивается во внешнем регистре адреса (обычно вместе с *A*19-*A*16 по стробу *ALE*) и используется как дополнительный адресный выход, определяющий доступ к старшему банку памяти либо к ВУ с байтовой организацией, подключенному к старшей половине шины *AD*. Совместное использование *BHE* и младшей линии адреса *A*0 для дешифрации адресов позволяет осуществлять передачу слов или отдельных байтов по шине *AD* (табл.3). Отметим, что после окончания сигнала *BHE* на выход подается резервный сигнал состояния *ST*7, не имеющий определенного значения.

Таблица 3

BHE *A*0	Разрядность данных
0 0	16-разр. слово (оба байта)
0 1	Старший байт *D*15-*D*8, нечетный адрес
1 0	Младший байт *D*7-*D*0, четный адрес
1 1	Нет обращения

ALE – строб адреса (разрешение защелкивания адреса), выдается в начале каждого цикла шины и используется для записи адреса и сигнала *BHE* в регистры-защелки, т.е. для демультиплексирования шины *AD*.

DEN (или **DE**) – строб данных (разрешение передачи данных). Выдается в циклах чтения, записи для разрешения выхода шинных формирователей.

RD – чтение, идентифицирует выполнение цикла чтения из ЗУ или ВУ (в зависимости от значения сигнала *M*/*IO*). Указывает этим устройствам на необходимость выдачи данных на шину.

WR – запись, указывает на выполнение цикла записи в ЗУ или ВУ и сопровождает данные, выдаваемые микропроцессором на шину.

M/IO – является признаком обращения к ЗУ (*M*/*IO*=1) или ВУ (*M*/*IO*=0) и используется для разделения адресного пространства памяти и ввода-вывода. Значение *M*/*IO* = 0 появляется только при выполнении портовых команд ввода (*IN*) и вывода (*OUT*).

DT/R – передача/прием данных, определяет направление передачи по шине *AD*: *DT*/*R*=1 – запись данных из МП в ЗУ или ВУ, *DT*/*R*=0 – чтение данных из ЗУ или ВУ в МП. Предназначен для управления шинными формирователями и действует на протяжении всего цикла шины, как и сигнал *M*/*IO*. Отметим, что направление передачи данных через шинные формирователи может также определяться с помощью сигналов *RD* и *WR*, но они имеют меньшую длительность и поэтому менее удобны для «медлительных» шинных формирователей.

HOLD – запрос шины (запрос захвата) от внешней подсистемы (ВУ или контроллера прямого доступа к памяти i8237).

HLDA – подтверждение захвата шины, выдается в ответ на сигнал *HOLD* после приостанова вычислительного процесса в МП и перевода шины *AD* и некоторых управляющих сигналов в *z*-состояние. При *HLDA*=1 подсистема, инициирующая запрос захвата, может использовать шину самостоятельно. После установления *HOLD*=0 ЦП выдает сигнал *HLDA*=0, возобновляет управление шиной и продолжает работу по программе.

NMI – немаскируемое прерывание, распознается микропроцессором по завершению текущей команды независимо от состояния флага разрешения прерывания *IF*. Этот вход предназначен для сигнализации о некоторых критических ситуациях, например об аварийном отключении сетевого питания.

INTR – запрос прерывания (маскируемый), опрашивается процессором в конце выполнения каждой команды, если прерывания разрешены (*IF*=1), то запрос фиксируется во внутреннем триггере и обслуживается. Обычно на вход *INTR* подается запрос от программируемого контроллера прерываний i8259A. Если *IF*=0, то запрос по входу *INTR* игнорируется.

INTA – подтверждение запроса прерывания, формируется в ответ на принятый запрос прерывания *INTR*, выполняет функцию сигнала *RD* в цикле подтверждения прерывания и стробирует считывание указателя адреса (вектора) прерывания. В каждом случае подтверждения прерывания выполняются два цикла *INTA*, из которых первый является предварительным и не сопровождается чтением информации.

READY – готовность, указывает на то, что адресуемое в данном цикле устройство готово к обмену данными. Если устройство не готово к взаимодействию с МП, оно выдает сигнал *Ready*=0, и МП переходит в состояние ожидания. В этом случае между тактами *T*3 и *T*4 цикла шины (перед завершением цикла) появляется необходимое число тактов ожидания *TW*. После установки сигнала *Ready*=1 МП выходит из состояния ожидания и возобновляет работу.

TEST – проверка, используется вместе с командой ожидания *WAIT*, выполняя которую

МП проверяет уровень сигнала *TEST*. Если *TEST*=0, МП переходит к выполнению следующей по порядку команды. Если *TEST*=1, МП вводит холостые такты *TI* и периодически, с интервалом 5*T*, проверяет значение сигнала *TEST*. Команда *WAIT* и сигнал *TEST* обеспечивают синхронизацию работы МП с внешними сигналами: *TEST* – вход программной проверки, *RDY* – вход аппаратной проверки готовности устройств в системе.

CLK – тактовая синхронизация (тактирование). Сигнал синхронизации от внешнего генератора тактовых импульсов, предназначен для синхронизации МП. Используется серия тактовых импульсов *CLK* с периодом повторения *T*, равным 100-500 нс (модели разных годов).

RESET – сброс, переводит МП в определенное начальное состояние, в котором сброшены сегментные регистры (кроме *CS*, все разряды которого устанавливаются в единичное состояние), указатель команд *IP*, все флаги, регистры очереди команд и все внутренние триггеры в устройстве управления. Сигнал *RESET* не влияет на состояние общих регистров, которые устанавливаются в начальное состояние программным путем. На время действия сигнала *RESET* все выходы, имеющие три состояния, переводятся в третье состояние, а выходы, имеющие два состояния, становятся пассивными. Минимальная продолжительность сигнала *RESET* при первом включении МП составляет 50 мкс, а при повторном запуске – четыре такта синхронизации. После снятия сигнала *RESET* работа МП возобновляется из начального состояния.

MN/MX – минимальный/максимальный режимы. Сигнал на этом входе определяет режим работы МП: 1 -минимальный, 0 -максимальный, когда изменяются функции восьми управляющих сигналов.

В максимальном режиме действуют следующие управляющие сигналы.

ST2-ST0 – сигналы состояния, обеспечивающие информацию о типе выполняемого цикла шины (табл. 4). Сигналы состояния подаются в контроллер шины, который дешифрует их и формирует расширенный набор управляющих сигналов. Если МП не инициирует цикл шины, то сигналы *ST*2-*ST*0 устанавливаются в пассивное состояние: 111. Отметим, что сигнал *ST*2 логически эквивалентен сигналу *M*/*IO*, а *ST*1 – сигналу *DT*/*R*.

Таблица 4

*ST*2 *ST*1 *ST*0	Тип цикла шины
0 0 0	Подтверждение прерывания
0 0 1	Чтение ВУ (чтение порта)
0 1 0	Запись ВУ (запись в порт)
0 1 1	Останов
1 0 0	Выборка команды
1 0 1	Чтение ЗУ (загрузка из памяти)
1 1 0	Запись ЗУ (сохранение в память)
1 1 1	Цикла шины нет

QS1, **QS0** – состояние очереди. Идентифицирует состояние внутренней 6-байтовой очереди команд МП (табл. 5) и действует в течение такта синхронизации после выполнения операции над очередью. Сигналы *QS*1, *QS*0 предназначены для сопроцессора, который воспринимает команды и операнды с помощью команды *ESC*. Сопроцессор контролирует шину *AD* и фиксирует момент, когда из программной памяти выбирается предназначенная для него команда *ESC*, а затем следит за очередью команд и определяет момент, когда эта команда должна выполняться.

Таблица 5

*QS*1 *QS*0	Операции над очередью
0 0	Операции нет, в последнем такте не было выборки из очереди
0 1	Из очереди выбран первый байт команд
1 0	Очередь пуста, была опустошена командой передачи управления
1 1	Из очереди выбран следующий байт команды

RQ/GT1, **RQ/GT0** – запрос/представление (подтверждение, разрешение). Две

одинаковые двунаправленные линии, каждая из которых может использоваться для передачи импульсных сигналов запроса/разрешения доступа к локальной шине (каналу). Процесс доступа к шине осуществляется в следующем порядке:

1) устройство, подключенное к локальной шине и требующее доступа к общим ресурсам, формирует запросный (первый) импульс длительностью один такт;
2) в конце текущего цикла МП выдает ответный (второй) импульс, подтверждающий возможность доступа к локальной шине (в следующем такте МП переводит шины адреса/данных и управления в высокоомное состояние и отключается от канала);
3) по окончании работы с каналом устройство выдает на ту же линию импульс (третий), указывающий на окончание захвата канала (в следующем такте МП возобновляет управление шиной и продолжает вычисления).

Все три импульса имеют одинаковую длительность и низкий активный уровень. Сигналы на линиях независимы, однако линия *RQ*/*GT*0 имеет более высокий приоритет, чем линия *RQ*/*GT*1, когда запросы поступают одновременно. Но если на линии *RQ*/*GT*0 появляется запрос в то время, когда МП находится в состоянии захвата по сигналу *RQ*/*GT*1, то этот запрос захвата не получает подтверждения до освобождения шины по линии *RQ*/*GT*1. Таким образом, каждая из двух рассмотренных линий служит для установления режима захвата шин и в этом отношении эквивалентна паре линий *HOLD* и *HLDA* МП ВМ86 в минимальном режиме.

LOCK – блокировка шины, информирует устройства системы, что они не должны пытаться запрашивать шину. Формируется однобайтовым префиксом *LOCK*, располагаемым перед командой, и действует до конца выполнения этой команды, запрещая доступ к системной магистрали другим устройствам, в частности другим процессорам. При подтверждении запроса шины выходной буфер сигнала *LOCK* переводится в третье состояние.

Префикс *LOCK* не влияет на прерывания. Если при наличии блокировки внешняя система запрашивает шину по линиям *RQ*/*GT*, МП фиксирует запрос, но не подтверждает его до завершения команды, имеющей префикс блокировки. Программисты обычно используют этот префикс, когда необходимо идентифицировать состояние разделяемых ресурсов системы. Префикс *LOCK* может использоваться и в минимальном режиме, когда внешний сигнал блокировки *LOCK* отсутствует. В этом случае генерирование подтверждения *HLDA* на запрос шины *HLD* задерживается до завершения выполняемой команды.

2.1 ОПИСАНИЕ ВНУТРЕННЕЙ АРХИТЕКТУРЫ

Микропроцессор К1810ВМ86 (i8086) содержит 14 16-разрядных внутренних регистров и образует 16-разрядную шину данных для связи с внешней памятью и портами ввода-вывода. Шина адреса имеет 20 линий, что позволяет непосредственно адресовать память емкостью до 2^{20} = 1 048 576 байт (1 Мбайт).

Поскольку разрядность регистров МП 16 бит, а адресация 20-разрядная, то необходим механизм вычисления адреса. Таким механизмом стала сегментация памяти – разделение пространства памяти на перекрывающиеся сегменты (участки), каждый из которых имеет размер 64 Кбайт (2^{16}=65536) и начинается на 16-байтной границе (называемой границей параграфа). Причем в любой момент времени МП может обращаться к ячейкам четырех сегментов, которые программно выбраны в качестве текущих. Сегментация памяти обеспечивает удобный механизм вычисления физических адресов и способствует модульному проектированию программного обеспечения, что упрощает программирование и отладку.

Структурная схема МП ВМ86 (рис.6) содержит две относительно независимые части:

1) операционное устройство, реализующее заданные командой операции;
2) устройство шинного интерфейса, осуществляющее выборку команд из памяти, а также обращение к памяти и внешним устройствам для считывания операндов и записи результатов.

Оба устройства могут работать параллельно, что обеспечивает совмещение во времени процессов выборки и исполнения команд. Это повышает быстродействие МП, так как операционное устройство, как правило, выполняет команды (длина команд МП варьируется от 1 до 6 байт), коды которых уже находятся в МП, и поэтому такты выборки команды не включаются в ее цикл.

1. Операционное устройство МП содержит группу общих регистров, арифметико-логическое устройство (АЛУ), регистр флагов *F*.

1.1. Восемь 16-разрядных регистров:

1) *AX, BX, CX, DX* используются прежде всего для хранения данных, особенностью этих регистров является то, что они допускают раздельное использование их младших байтов *AL, BL, CL, DL* и старших байтов *AH, BH, CH, DH*;
2) регистры *SP, BP, SI, DI* хранят главным образом адресную информацию.

При обращении к этой группе регистров их адрес кодируется непосредственно в команде в виде трехбитового кода.

1.2. Арифметико-логическое устройство содержит

- 16-разрядный комбинационный сумматор, с помощью которого выполняются арифметические операции;
- наборы комбинационных схем для выполнения логических операций, схемы для операций сдвигов и десятичной коррекции;
- регистры для временного хранения операндов и результатов.

1.3. Регистр флагов *F*, шесть арифметических флагов которого фиксируют определенные признаки результата выполнения операции. Значения этих флагов (кроме флага *AF*) используются для реализации условных переходов, изменяющих ход выполнения программы. Различные команды влияют на флаги по-разному. Назначение арифметических флагов:

CF (бит 0)	флаг переноса, фиксирует значение переноса (заема), возникающего при сложении (вычитании) байтов или слов, а также значение выдвигаемого бита при операциях сдвига операнда
PF (бит 2)	флаг четности (или паритета), равен 1 если результат операции содержит четное число единиц в младшем байте
AF (бит 4)	флаг вспомогательного переноса, фиксирует перенос (заем) из младшей тетрады (из бита *b*3) в старшую при сложении (вычитании), используется только для двоично-десятичной арифметики, которая работает только с младшими байтами
ZF (бит 6)	флаг нуля, устанавливается в 1 если результат операции равен нулю
SF (бит 7)	флаг знака, дублирует значение старшего бита результата, который при использовании дополнительного кода соответствует знаку числа
TF (бит 8)	флаг трассировки, при *TF* =1 МП переходит в покомандный (пошаговый) режим работы, применяемый при отладке программ, когда автоматически генерируется сигнал внутреннего прерывания после выполнения каждой команды с целью перехода к соответствующей подпрограмме
IF (бит 9)	флаг разрешения прерываний, управляемый с помощью команд *CLI* и *STI*; при *IF*=1 МП воспринимает (распознает) и соответственно реагирует на запрос прерывания по входу *INTR*; при *IF*=0 прерывания по этому входу запрещаются (маскируются) и МП игнорирует поступающие запросы прерываний (значение флага не влияет на восприятие немаскируемых прерываний по входу *NMI*, а также программных прерываний, выполняемых по команде *INT*)
DF (бит 10)	флаг направления, управляемый командами *CLD* и *STD*, определяет порядок обработки цепочек (строк) в соответствующих командах: от меньших адресов к большим (*DF*=0) или наоборот (*DF*=1)
OF (бит 11)	флаг переполнения, сигнализирует о потере старшего бита результата сложения или вычитания в связи с переполнением разрядной сетки при работе со знаковыми числами (при сложении этот флаг устанавливается в 1, если происходит перенос в старший бит и нет переноса из старшего бита или имеется перенос из старшего бита, но отсутствует перенос в него; в противном случае флаг устанавливается в 0; при вычитании он устанавливается в 1, когда возникает заем из старшего бита, но заем в старший бит отсутствует либо имеется заем в старший бит, но отсутствует заем из него)

2. Устройство шинного интерфейса (или просто шинный интерфейс) содержит очередь команд, буферы (обеспечивающие связь с шиной), блок сегментных регистров, сумматор адресов и указатель команд. Шинный интерфейс выполняет операции обмена между МП и памятью или портами ввода-вывода по запросам операционного устройства. Когда операционное устройство занято выполнением команды, шинный интерфейс самостоятельно инициирует опережающую выборку кодов очередных команд из памяти.

2.1. Очередь команд представляет собой набор байтовых регистров и выполняет роль регистра команд, в котором хранятся коды, выбранные из памяти. Длина очереди составляет 6 байт, что соответствует максимально длинному формату команд. Наличие очереди команд, а также способность операционного устройства и шинного интерфейса работать параллельно позволяют совместить во времени фазы выборки команды и выполнения заданной операции: пока одна команда исполняется в операционном устройстве, шинный интерфейс осуществляет выборку следующей команды. Таким образом, достигаются высокая плотность загрузки шины и повышение скорости выполнения программы. Пример, иллюстрирующий реализацию описанного конвейерного принципа, приведен на рис.8, где *TI* обозначает холостые такты работы шины, когда очередь команд заполнена, а операционное устройство занято выполнением текущей команды и не запрашивает выполнения цикла шины.

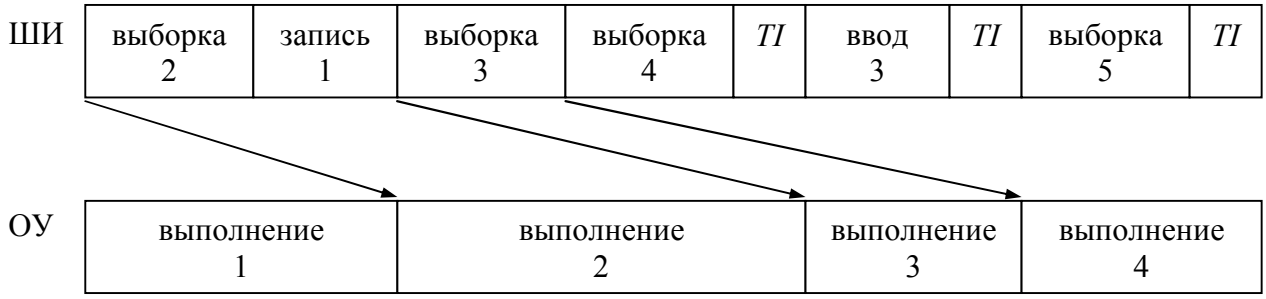


Рис. 8 – Пример конвейерного выполнения команд (фрагмент)

Шинный интерфейс инициирует выборку следующего командного слова автоматически, как только в очереди освободятся два байта. Как правило, в очереди находится минимум один байт потока команд, так что операционное устройство не ожидает выборки команды. Ясно, что опережающая выборка команд позволяет экономить время только при естественном порядке выполнения команд. Когда операционное устройство выполняет команду передачи управления (перехода) в программе, шинный интерфейс сбрасывает очередь, выбирает команду по новому адресу, передает ее в операционное устройство, а затем начинает заполнение очереди из следующих ячеек памяти (реинициализацию). Эти действия предпринимаются в условных и безусловных переходах, вызовах подпрограмм, возвратах из подпрограмм и при обработке прерываний.

По мере необходимости операционное устройство считывает байт из очереди и выполняет предписанную командой операцию. При многобайтовых командах из очереди считываются и другие байты команды. В тех редких случаях, когда к моменту считывания очередь оказывается пустой, операционное устройство ожидает выборку очередного командного слова, которую инициирует шинный интерфейс. Если команда требует обращения к памяти или порту ввода-вывода, операционное устройство запрашивает шинный интерфейс на выполнение необходимого цикла шины для передачи данных. Когда шинный интерфейс не занят выборкой команды, он удовлетворяет запрос немедленно; в противном случае операционное устройство ожидает завершения текущего цикла шины. Со своей стороны, шинный интерфейс приостанавливает выборку команд во время обмена данными между операционным устройством и памятью или портами ввода-вывода.

2.2. Буфер шины адреса/данных (БАД) содержит 16 двунаправленных управляемых усилителей с тремя выходными состояниями и обеспечивает номинальную нагрузочную способность линий *AD*15-*AD*0.

Буфер шины адреса/состояния (БАС) содержит четыре однонаправленных усилителя с тремя выходными состояниями и обеспечивает номинальную нагрузочную способность линий

*A*19/*ST*6-*A*16/*ST*3.

2.3. Сегментные регистры хранят базовые (начальные) адреса сегментов памяти: кодового сегмента *CS* (*code segment*), в котором содержится программа; стекового сегмента *SS* (*stack segment*); сегмента данных *DS* (*data segment*); дополнительного сегмента *ES* (*external segment*), в котором обычно содержатся данные. Наличие сегментных регистров обусловлено разделением памяти на сегменты и используемым способом формирования адресов памяти. 20-разрядный физический адрес памяти образуется с помощью указания сегмента и внутрисегментного смещения. Внутрисегментное смещение может быть вычислено в соответствии с указанным в команде способом адресации, в частности, может находиться в формате самой команды или содержаться в регистре.

2.4. Сумматор адресов осуществляет вычисление 20-разрядных физических адресов (складывает значение из сегментного регистра, сдвинутое влево на 4 бита, со смещением).

2.5. Указатель команд *IP* хранит смещение следующей команды в текущем кодовом сегменте, т.е. указывает на следующую по порядку команду. Модификация *IP* осуществляется шинным интерфейсом так, что при обычной работе *IP* содержит смещение того командного слова, которое шинный интерфейс будет выбирать из памяти. Оно не совпадает со смещением очередной команды (находящейся в этот момент на выходе очереди команд), которую будет выполнять операционное устройство. Поэтому при запоминании содержимого *IP* в стеке, например при вызове подпрограмм, оно автоматически корректируется, чтобы адресовать следующую команду, которая будет выполняться. Эта особенность является следствием опережающей выборки команд, реализованной в МП. Непосредственный доступ к *IP* имеют команды передачи управления.

3. Управляющее устройство или устройство управления (УУ) дешифрует команды, а также воспринимает и вырабатывает необходимые управляющие сигналы. В его состав входит блок микропрограммного управления, в котором реализовано программирование МП на микрокомандном уровне. Система команд микропроцессора i8086 состоит из 98 команд: 19 команд передачи данных, 38 команд их обработки, 24 команд перехода и 17 команд управления процессором. Каждая команда состоит из кода операции (КОП), идентифицирующего её, и операндов, несущих требуемую для операции информацию. Команды могут содержать несколько операндов. Но чем больше операндов – тем команды длиннее, т.е. занимают больше места в памяти и тем больше времени требуется для передачи её в МП. КОП и операнды могут иметь произвольную длину и не обязаны быть непрерывными (в то же время общая длина команды должна быть целым числом байт).

Примеры КОП некоторых команд МП:

1) Загрузка в регистр AX константы (mov AX,1002h): 10111000 00000010 00010000
2) Команда сложения регистров вида Rg1=Rg1+Rg2: 000000dw 11 rg1 rg2
   бит *d* – направление (=1 –to, =0 -from);
   бит *w* – признак слова (=1 -16 бит, =0 -8бит)
   add AX,BX: 00000011 11 000 011
3) Команда сброса флага *Carry* (CLC): 11111000

Писать в машинных кодах (записывая последовательности КОП с данными, адресами) крайне неудобно. Поэтому машинным кодам поставили в соответствие «короткие слова» – мнемоники, отражающие суть структуры исполнительного узла (архитектуры МП). Множество мнемоник, образующих систему команд, с правилами оформления кода (семантикой) представляет собой язык программирования называемый ассемблером.

2.2 АДРЕСНОЕ ПРОСТРАНСТВО ПАМЯТИ И ПОРТОВ ВВОДА-ВЫВОДА

Размещение байтов и слов в памяти. Память логически организована как одномерный массив байтов, каждый из которых имеет 20-разрядный физический адрес в диапазоне 00000-FFFFFh. Любые два смежных байта в памяти могут рассматриваться как 16-битовое слово. Младший байт слова имеет меньший адрес, а старший – больший. Такое размещение байтов слова используется в большинстве ЭВМ. Адресом слова считается адрес его младшего байта. Таким образом, 20-разрядный адрес памяти может рассматриваться как адрес байта (ячейки) и как адрес слова (последовательное расположение двух байт), так и более сложных структур (массивов, записей и т.п.).

Полная информация, необходимая для определения физического адреса, содержится в адресном объекте «сегмент: смещение», который называется указателем адреса и содержит адрес сегмента (16 разрядов) и внутрисегментное смещение (16 разрядов). Для запоминания указателя адреса требуется два слова памяти, причем слово с меньшим адресом всегда содержит смещение, а слово с большим адресом – базовый адрес сегмента. Каждое слово хранится обычным образом, т.е. по принципу «младший байт – по меньшему адресу».

Команды, байты и слова данных можно свободно размещать по любому адресу, что позволяет экономить память благодаря ее плотной упаковке. Слово с четным адресом называется выровненным на границе слов. Слова с нечетными адресами (не выровненные) также допустимы, но для их передачи требуются два цикла шины, что снижает производительность МП. Отметим, что шинный интерфейс инициирует необходимое для выборки слова число обращений к памяти автоматически, так что двукратное обращение к памяти не требует специального указания в программе. Особенно важно иметь выровненные слова для операций со стеком, так как в них участвуют только слова. Поэтому, указатель стека *SP* необходимо всегда инициализировать на четный адрес.

Команды всегда выбираются словами по четным адресам, за исключением первой выборки после передачи управления по нечетному адресу, когда выбирается один байт. Поток команд разделяется на байты при заполнении очереди команд внутри МП, так что выравнивание команд не влияет на производительность и поэтому не используется.

Сегментация памяти и вычисление адресов. Пространство памяти емкостью 1 Мбайт представляется как набор сегментов, определяемых программным путем. Сегмент состоит из смежных ячеек памяти и является независимой и отдельно адресуемой единицей памяти емкостью 64 Кбайт. Каждому сегменту программой назначается начальный (базовый) адрес, являющийся адресом первого байта сегмента в пространстве памяти. Начальные адреса четырех сегментов, выбранных в качестве текущих, записываются в сегментные регистры *CS*, *DS*, *SS* и *ES*, тем самым фиксируются текущие сегменты кода (программы), данных, стека и дополнительных данных соответственно. Для обращения к командам и данным, находящимся в других сегментах, необходимо изменять содержимое сегментных регистров, что позволяет использовать все пространство памяти емкостью 1 Мбайт. Сегментные регистры инициализируются в начале программы путем записи в них соответствующих констант. Например: DS=0A00h, BX=0008h указывает что по адресу DS:BX хранится байт или слово или структура данных или массив (непрерывный участок памяти) байтов, слов и т.п. Пара 16-разрядных чисел, адресующих ячейку памяти или их непрерывную совокупность, называется логическим адресом. Другими словами, логический адрес ячейки памяти состоит из двух 16-разрядных беззнаковых значений: адреса сегмента (называется также просто базой или сегментом) и внутрисегментного смещения (определяет расстояние от начала сегмента до этой ячейки). Получая логический адрес, МП обязан выдать на шину физический адрес, представляющий собой 20-разрядное число в диапазоне 0-FFFFFh, которое однозначно определяет положение каждого байта в пространстве памяти емкостью 1 Мбайт. В начале каждого цикла шины, связанного с обращением к памяти, физический адрес выдается на шину адреса и сопровождается сигналом *ALE*. Для вычисления физического адреса значение сегмента сдвигается влево на 4 бита (умножается на 16) и суммируется со смещением, как показано на рис.9. Перенос из старшего бита, который может возникнуть при суммировании, игнорируется. Это приводит к так называемой кольцевой организации памяти, при которой за ячейкой с

максимальным адресом FFFFFh следует ячейка с нулевым адресом. Аналогичную кольцевую организацию имеет и каждый сегмент. Каждая физическая ячейка памяти может принадлежать одному или сразу нескольким сегментам.

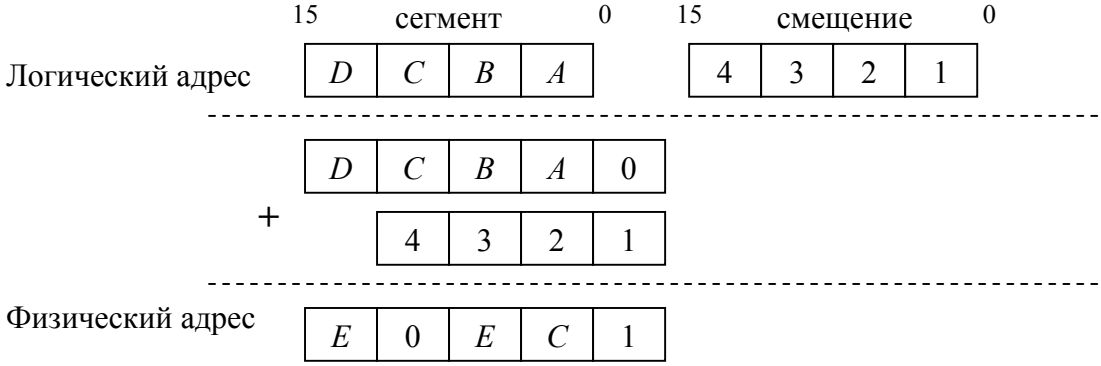


Рис. 9 – Вычисление физического адреса

Источники логического адреса для различных типов обращения к памяти приведены в табл.6:

- команды всегда выбираются из текущего сегмента кода в соответствии с логическим адресом *CS:IP*;
- стековые команды всегда обращаются к текущему сегменту стека по адресу *SS:SP*;
- если при вычислении адреса *EA* используется регистр *BP*, то обращение производится также к стековому сегменту, при этом принцип стека «первый пришел – последний вышел» игнорируется и ячейки стекового сегмента рассматриваются как ОЗУ с произвольной выборкой, что обеспечивает большую гибкость в использовании этих ячеек.

Таблица 6

Тип обращения к памяти	Сегмент (по умолчанию)	Вариант	Смещение
Выборка команды	*CS*	Нет	*IP*
Стековая операция	*SS*	Нет	*SP*
Переменная	*DS*	*CS, SS, ES*	*EA*
Цепочка-источник	*DS*	*CS, SS, ES*	*SI*
Цепочка-приемник	*ES*	Нет	*DI*
BP как базовый регистр	*SS*	*CS, ES, DS*	*EA*

Сегментная структура памяти обеспечивает возможность создания позиционно независимых или динамически перемещаемых программ, что необходимо в мультипрограммной среде для эффективного использования оперативной памяти. Чтобы обеспечить позиционную независимость, все смещения в программе должны задаваться относительно фиксированных значений, содержащихся в сегментных регистрах. Это позволяет произвольно перемещать программу в адресном пространстве памяти, изменяя только содержимое сегментных регистров.

Стек, как обычно, организуется в ОЗУ, и его положение определяется содержимым регистров *SS* и *SP*. Регистр *SS* хранит базовый адрес текущего сегмента стека, а регистр *SP* указывает на вершину стека, т.е. содержит смещение вершины стека в стековом сегменте. При каждом обращении к стеку пересылается одно слово, причем содержимое *SP* модифицируется автоматически: при записи (включении) в стек оно уменьшается на два, при чтении (извлечении) из стека – увеличивается на два.

Учитывая архитектуру МП компиляторы (в рекомендательном порядке) производят выравнивание кода, данных на границу слов (т.е. адреса четны), либо параграфа (на границу в 16 байт).

2.3 ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА

Для ввода-вывода применяются специальные команды *IN* (ввод) и *OUT* (вывод), которые обеспечивают передачу данных между аккумулятором *AL* или *AX* и адресуемыми портами. При выполнении этих команд вырабатывается сигнал *M*/*IO*=0, который идентифицирует выбор пространства ввода-вывода и в совокупности с сигналами *WR* и *RD* позволяет сформировать системные сигналы *IOW* и *IOR* для управления операциями записи данных в порт и чтения из порта. Команды *IN* и *OUT* могут использовать прямую адресацию (когда адрес порта содержится в виде константы во втором байте команды) и косвенную адресацию (когда адрес располагается в регистре *DX*). В первом случае можно адресовать до 256 портов для ввода и вывода данных. Во втором обеспечивается адресное пространство до 64К 8-битовых портов или до 32К 16-битовых портов. Косвенная адресация позволяет вычислять адреса портов при выполнении программы и удобна при организации вычислительных циклов для обслуживания нескольких портов с помощью одной процедуры.

Восемь ячеек F8-FFh в пространстве ввода-вывода зарезервированы для системных целей. Использовать их в прикладных программах не рекомендуется.

МП может передавать по шине байт или слово в/из ВУ. Чтобы слово передавалось за один цикл шины, адрес ВУ должен быть четным. Адрес байтового ВУ может быть четным или нечетным, и соответственно порты этих внешних устройств подключаются к линиям младшего и старшего байта шины данных. Для раздельного обращения к этим портам дешифрирование адресов осуществляется с учетом сигналов на линиях *BHE* и *A*0.

Порты ввода-вывода применялись для подключения устройств хранения на магнитных дисках, обращения с видеопроцессором (графической картой). На сегодняшний день интерфейсы *IDE*, *SATA* используют портовые операции для "общения" с жесткими дисками (*HDD*), *flash* дисками, оптическими системами хранения данных и т.п.

2.4 ЦИКЛ МИКРОПРОЦЕССОРА ВМ86

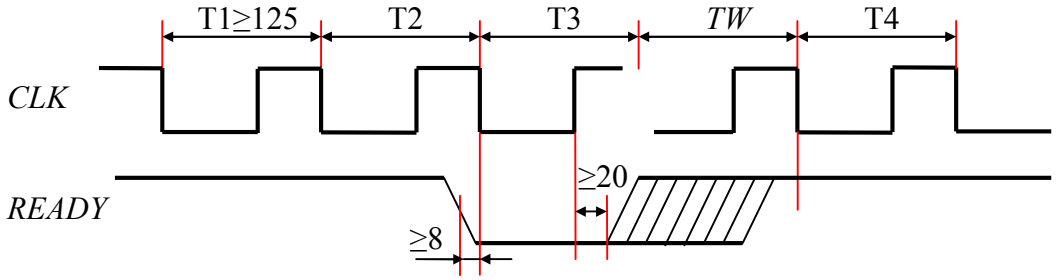
Выполнение команд можно представить последовательностью циклов шины (циклов обмена), в течение которых МП обращается к памяти за командами или обменивается данными с памятью или внешними устройствами. Каждый цикл шины инициируется устройством шинного интерфейса и содержит четыре обязательных такта *Т*1-*Т*4.

В такте *Т*1 выдается адрес на совмещенную шину адреса/данных, в такте *Т*2 производится коммутация направления передачи, в тактах *Т*3 и *Т*4 – передача данных.

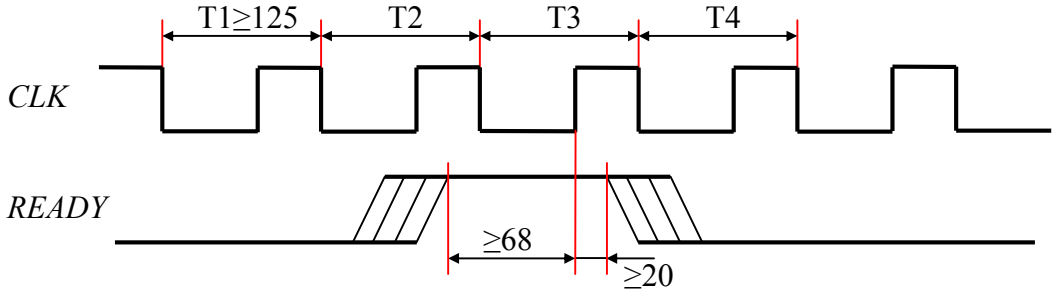
Если в системе имеются устройства, которые не могут передавать информацию с максимальной пропускной способностью шины, то с помощью сигнала готовности *READY* вводится необходимое число тактов ожидания *TW* между тактами *Т*3 и *Т*4. В зависимости от выбранного способа управления сигналом *READY* система может быть выполнена в виде «нормально готовой» или «нормально неготовой».

В варианте «нормально готовой» системы устройства с недостаточным быстродействием должны установить сигнал *READY* =0 не позднее чем за 8 нс до начала такта *Т*3 (рис.10а) и удерживать его активным не менее 20 нс после переднего фронта такта *Т*3. Такты ожидания *TW* будут добавлены, пока сигнал *READY* активен. Ближайший задний фронт такта ожидания при *READY*=1 приведет к завершению такта *Т*3.

В «нормально неготовой» системе устройства обычно требуют введения тактов ожидания *TW*. Если же выбрано устройство, не нуждающееся в этом, то оно должно своевременно (более чем за 68 нс до переднего фронта такта *Т*3) обеспечивать *READY*=1, чтобы после *Т*3 наступил такт *Т*4 (рис.10б), т.е. предотвратить переход МП в состояние ожидания.



а) низкий уровень *READY* – требуются такты ожидания



б) высокий уровень *READY* – такты ожидания не нужны

Рис. 10 – Временные диаграммы сигнала *RDY* в "нормально готовой" (а) и "нормально неготовой" (б) системах

Цикл шины выполняется, когда требуется заполнить очередь команд или осуществить обмен данными в процессе выполнения команды. Если цикл шины не требуется, то формируются холостые такты *TW*, во время которых устройство шинного интерфейса остается пассивным. В течение такта *TW* на линиях *ST*6-*ST*3 МП сохраняет сигналы состояния от предыдущего цикла шины. Если в предыдущем цикле производилась запись, МП сохраняет на линиях *AD* записываемые данные до следующего цикла шины; если производилось чтение, МП не управляет линиями до начала следующего цикла шины. Число холостых тактов зависит от

длительности выполняемой команды и может быть достаточно большим. Так, для команды умножения байтов, содержащихся в регистрах, число тактов *TW* составляет 52÷69.

В *минимальном режиме* функционирование МП ВМ86 иллюстрируется временными диаграммами, приведенными на рис.11 (временные интервалы в нс).

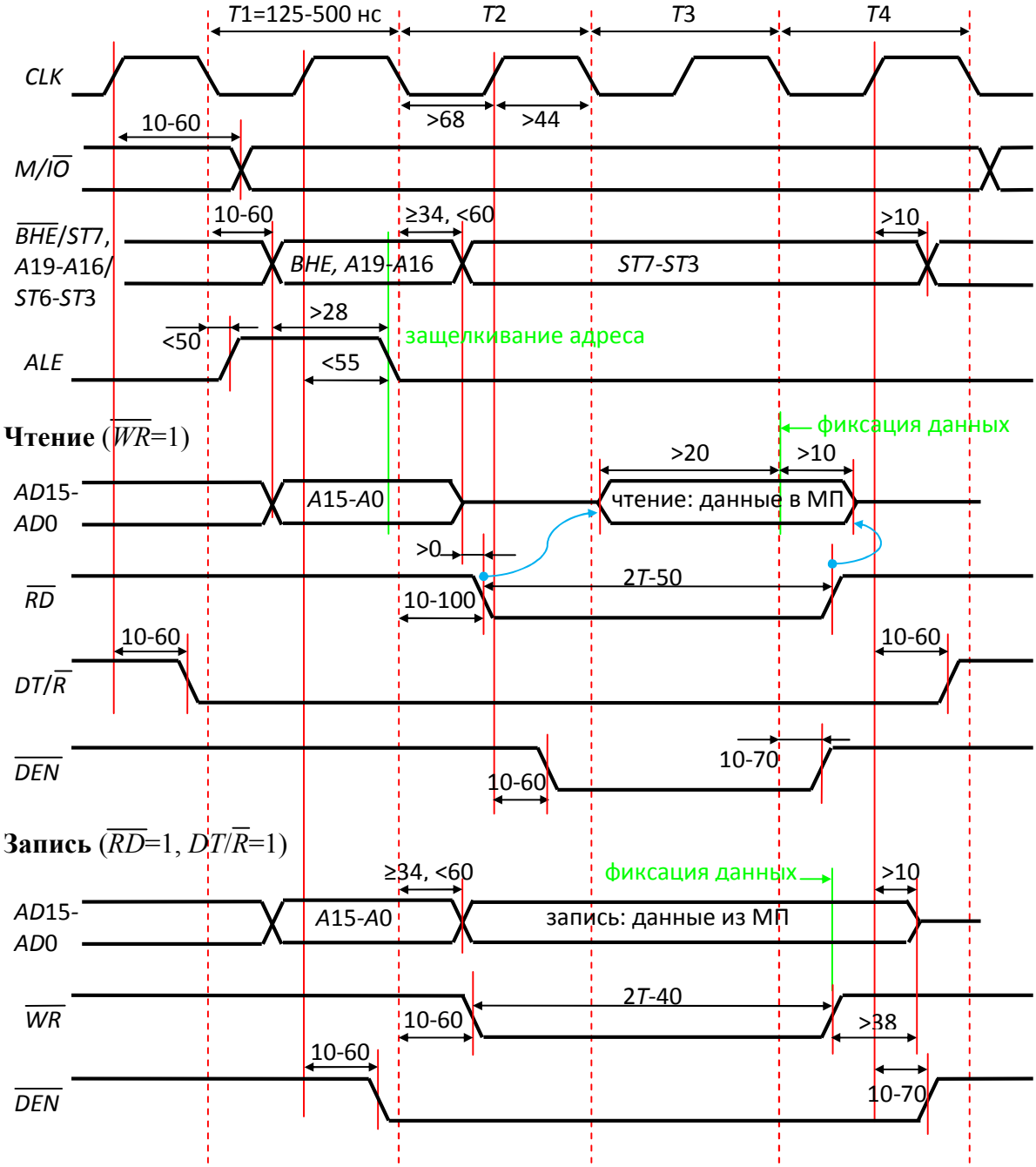


Рис. 11 – Временные диаграммы работы ВМ86 в минимальном режиме (F_{clk}≤8 МГц)

Цикл начинается с формирования в такте *T*1 сигнала *M*/*IO* определяющего тип устройства (ЗУ или ВУ), к которому производится обращение для пересылки данных. Длительность сигнала *M*/*IO* равна длительности цикла шины и он используется для селекции адреса устройства. В такте *T*1 и начале такта *T*2 МП выставляет адрес устройства на линиях *A*19-*A*16 и *AD*15-*AD*0, а также вырабатывает сигнал *BHE*, который вместе с *A*0 определяет передачу слова или одного из байтов. Одновременно с этим МП выдает строб адреса *ALE*, по спаду которого адрес фиксируется во внешних регистрах-защелках. На выходах этих регистров адрес сохраняется в течение всего цикла шины (до записи нового значения).

В такте *T*2 происходит переключение шин: на линии *A*19/*ST*6-*A*16/*ST*3 поступают сигналы состояния *ST*6-*ST*3, которые сохраняются до конца такта *T*4.

В цикле чтения в такте *T*2 линии *AD*15-*AD*0 переводятся в третье состояние до появления данных от адресованного устройства. В тактах *T*2-*T*4 вырабатывается сигнал чтения *RD*=0, который указывает этому устройству на необходимость выдачи данных. Для управления буферами данных, которые обычно подключаются к линиям *AD*15-*AD*0, в тактах *T*2-*T*4 формируется сигнал *DEN*, разрешающий передачу данных. Направление передачи данных при чтении определяет сигнал *DT*/*R*=0, действующий в течение всего цикла. После выполнения чтения и установления сигнала *RD*=1 микропроцессор заканчивает такт *T*4 следующим образом: линии *AD*15-*AD*0 находятся в высокоомном состоянии, сигналы *M*/*IO*, *DEN*, *DT*/*R*, *ST*7-*ST*3 неактивны, буферы данных отключены от канала.

В цикле записи в такте *T*2 адрес на линиях *AD*15-*AD*0 заменяется данными, предназначенными для записи в адресуемое устройство. Данные остаются действительными до середины такта *T*4 и сопровождаются сигналом *WR*=0, который используется в качестве строба для записи данных в устройство. Сигнал *DEN*=0 появляется уже в такте *T*1 и используется для подготовки буферов к передаче данных. Сигнал *DT*/*R*=1, переключающий буферы на передачу данных в направлении от МП, удерживается на протяжении всего цикла записи. После установления *WR*=1 такт *T*4 заканчивается так же, как и при выполнении цикла чтения.

Таким образом, циклы чтения и записи различаются не только активными значениями сигналов *RD* или *WR* и состоянием сигнала *DT*/*R,* но и тем, что в цикле записи сигналы *DEN* и *WR* становятся активными раньше и имеют большую длительность, чем в цикле чтения. Соответственно данные при записи присутствуют на шине в течение большего промежутка времени, чем при чтении.

В *максимальном режиме* работа шины адреса/данных МП ВМ86 эквивалентна работе в минимальном режиме. Как уже отмечалось, в максимальном режиме изменяется назначение восьми управляющих сигналов, в частности МП вырабатывает сигналы состояния *ST*2-*ST*0 (см. табл. 4), на основе которых системный контроллер КР1810ВГ88 формирует необходимые системные управляющие сигналы. Таким образом, МП в этом режиме работает обязательно совместно с системным контроллером, что отражено на временных диаграммах цикла шины (рис.12).

Код состояния *ST*2-*ST*0 выдается по срезу синхроимпульса в последнем такте предшествующего цикла. В такте *T*1 контроллер формирует строб *ALE* и устанавливает необходимый уровень сигнала на выходе *DT*/*R*. В такте *T*2 начинается формирование сигнала разрешения данных *DEN*, который в отличие от минимального режима имеет активный высокий уровень, а также следующих управляющих сигналов: *RD*, *MRDC* (чтение ЗУ), *IORC* (чтение ВУ), *AMWC* (опережающая запись ЗУ), *AIOWC* (опережающая запись ВУ). Для цикла записи в такте *T*3 начинается выработка сигналов *MWTC* (запись ЗУ, запись ВУ). В такте *T*4 цикл шины заканчивается: линии *AD*15-*AD*0 переводятся в третье состояние, устанавливаются сигналы *ST*2=*ST*1=*ST*0=1 и прекращается активное состояние управляющих сигналов. Необходимо отметить, что наличие специализированных сигналов чтения *MRDC* и *IORC* ставит под сомнение целесообразность использования общего сигнала чтения *RD*, тем более что последний требует буферизации. Отметим также, что опережающие сигналы записи, обеспечивающие более длительный импульс записи, требовались для некоторых типов БИС статической памяти и внешних устройств.

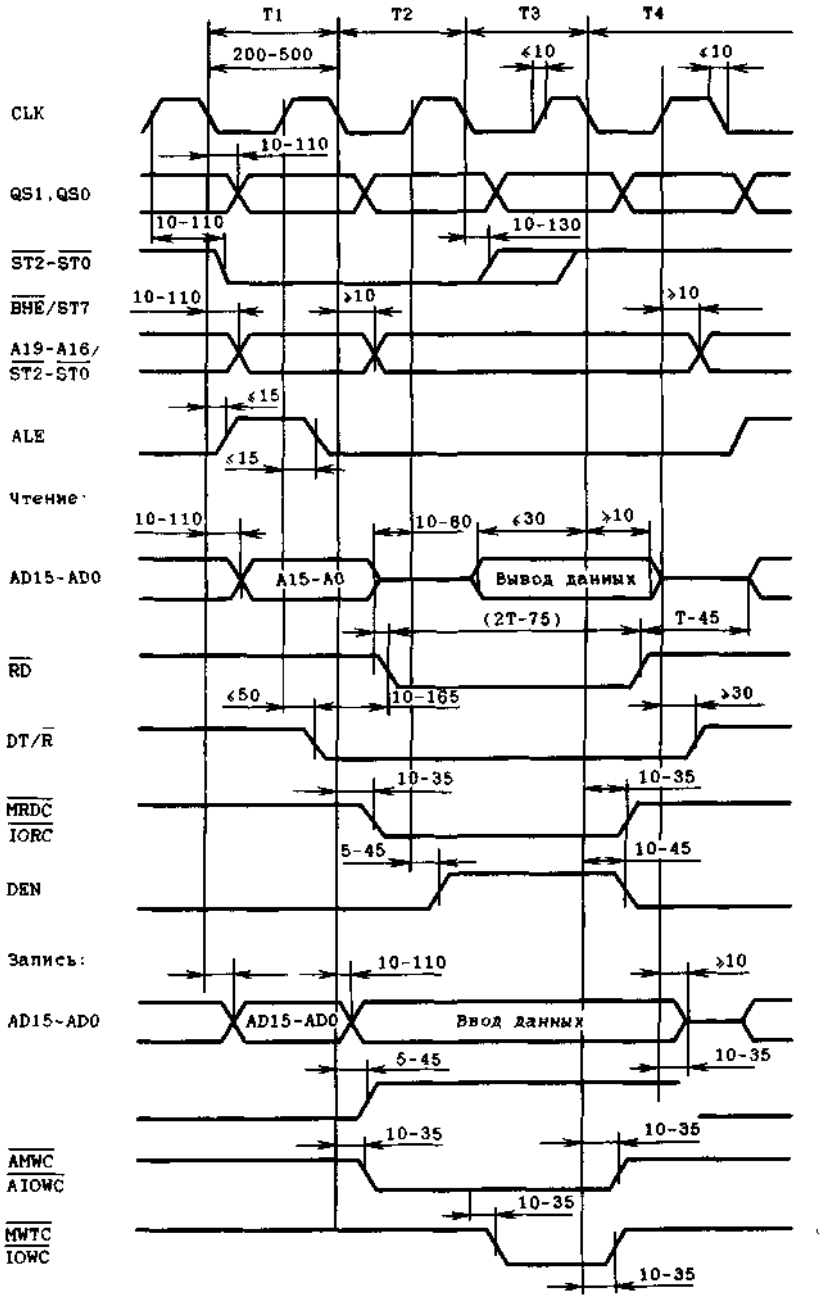


Рис. 12 – Временные диаграммы работы МП ВМ86 в максимальном режиме (F_{clk}≤5 МГц)

2.5 СИГНАЛ *RESET* ПРОЦЕССОРА

Активный уровень сигнала *Reset* переводит все выходные линии процессора в неактивное состояние (третье состояние), процессор очищает регистр флагов, перестает управлять локальной шиной, перестает выполнять текущие инструкции. По окончанию сигнала сброса процессор считывает конфигурационную информацию с некоторых входных линий и начинает свою работу. В более старших моделях таким образом устанавливаются коэффициент умножения тактовой частоты, режим работы кэша, роль процессора в многопроцессорной системе, способ подачи сигналов прерывания и т.п. Также если во время окончания сигнала *Reset* определенный вход процессора удерживать на низком уровне, то процессор начнет выполнение встроенного самотестирования, результат которого будет помещен в регистр-аккумулятор. Аппаратный сброс перезагружает регистры сопроцессора, аннулирует строки кэш-памяти, буферов трансляции и таблиц переходов.

Процессор начинает работу в реальном режиме, загружая при этом нули во все сегментные регистры за исключением *CS*, в который помещается значение F000h. В регистр *IP* загружается значение FFF0h. Таким образом, после снятия сигнала *Reset* процессор начинает выполнение инструкции из памяти по адресу FFFF0h. Поэтому в этой области адресов должно размещаться ПЗУ (*BIOS*), где обычно располагается инструкция (команда) дальнего перехода (*jmp seg:ofs*) – на программу загрузки ОС (т.к. 16 байт недостаточно для написания загрузчика ОС с внешнего носителя, ее инициализации, настройки и передачи ей управления). Сама программа загрузки также хранится в ПЗУ (*flash* памяти).

2.6 СИСТЕМА КОМАНД I8086

Все команды i8086 МП можно классифицировать в соответствии с рис.13.

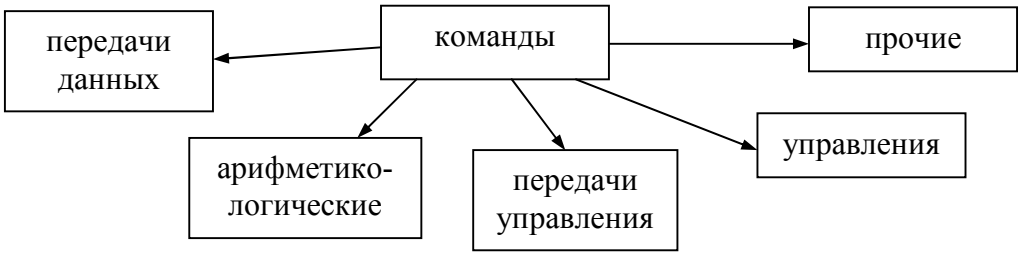


Рис. 13 – Классификация команд i8086

1. **Команды передачи данных** можно подразделить на

1.1) команды загрузки/сохранения регистров (присвоение значения второго операнда первому)

mov RG, const //RG=const
mov RG1, RG2 //RG1=RG2 (один из регистров может быть сегментным)
lahf/sahf – загрузка/сохранение в/из AH младшего байта регистра флагов
//прямая адресация, mem -имя переменной, объявленной в сегменте данных (здесь в качестве RG может выступать сегментный регистр)
mov RG, mem //RG=mem
mov mem, RG //mem=RG
//косвенная адресация, Ri -регистр, содержащий адрес переменной var
mov RG, [Ri] //RG=var
mov [Ri], RG //var=RG

Пример инициализации переменной var1 (объявлена в сегменте данных типа *word*):

```
mov AX, 2
// прямая адресация переменной (адрес содержится в команде)
mov var1, AX          // результат: var1=2
// косвенная адресация
mov BX, offset var1 // BX=адрес переменной (смещение), DS:BX -указатель на var1
mov DS:[BX], AX     // результат: var1=2
```

1.2) команды сохранения/загрузки значений регистров в стек:

- push/pop, операнд RG (в том числе сегментные) или mem
- pushf/popf – в качестве операнда выступает регистр флагов

Пример: push AX - выполняет: SS:[SP]=AX и SP=SP-2
pop AX - выполняет: SP=SP+2 и AX=SS:[SP]

Пример обмена значениями регистров AX и BX

```
push AX
mov AX, BX
pop BX
```

1.3) команды чтения/записи портов:

in RG,port/out port,RG (команды работают только с регистром AX если порт 16 разрядный и AL если 8-разрядный), если адрес порта больше 255 – необходимо использовать косвенную адресацию с регистром DX

Пример чтения слова из порта F0h и записи младшего байта этого слова в порт 0200h

```
in AX, F0h
mov DX, 200h
out DX, AL
```

1.4) строковые команды пересылки:

- movsb (movsw) – пересылка байта (слова) из адреса ds:[si] в адрес es:[di] с одновременным инкрементом или декрементом si, di на 1 (2) в соответствии с флагом направления (DF);
- lodsb (lodsw) – загрузка регистра al (ax) байтом (словом) из памяти с адресом ds:[si] с инкрементом или декрементом si на 1 (2) в соответствии с флагом направления (DF);

- stosb (stows) – сохранение значения регистра al (ax) в памяти по адресу es:[di] с инкрементом или декрементом di на 1 (2) в соответствии с флагом направления (DF);

Пример копирования массива байт mas1 в массив mas2 (объявлены в сегменте данных)

```
mov SI, offset mas1
mov DI, offset mas2
push DS
pop ES          // ES=DS
cld      // сбросить флаг направления (DF=0) – рост адресов
mov CX, nn    //количество пересылаемых байт, 1<=nn<=65535
rep movsb      //пересылка блока данных,  rep – префикс повторения CX раз
```

1.5) команда обмена значений регистров: xchg RG1,RG2; xchg RG,mem

1.6) команда загрузки эффективного адреса: lea RG, <имя переменной или метки> загружает регистровую пару DS:RG полным (логическим) адресом указанной переменной или метки.
lds RG, mem – загрузка логического адреса из памяти (указатель): RG=[mem], DS=[mem+2]
les RG, mem – загрузка логического адреса из памяти (указатель): RG=[mem], ES=[mem+2].
Пример формирования логического адреса переменной sum (объявлена в любом месте)

```
lea SI, sum          //load effective address, DS:SI – указывают на переменную sum
```

2. **Команды передачи управления** (ветвления) можно подразделить на

2.1) команды безусловной передачи управления:

- jmp <имя метки> – поместить в cs:[ip] адрес указанной метки (следующей за ней команды)
- call <имя метки> – вызов подпрограммы, команда сохраняет в стеке адрес точки возврата (адрес следующей команды) и выполняет jmp <имя метки>
- ret n – команда возврата из процедуры, извлекает из стека адрес точки возврата и передает управление команде с этим адресом, если присутствует константа n, то после извлечения точки возврата выполняет SP=SP+n (удаляет фактические параметры процедуры);
- int <номер прерывания> – вызывает программное прерывание, команда аналогична call, но в стеке сохраняется еще регистр флагов, а адрес процедуры извлекается из таблицы векторов прерываний (по умолчанию, располагается в первом килобайте ОЗУ);
- iret – возврат из прерывания, см. ret, но еще восстанавливает из стека регистр флагов.

Пример вызова процедуры

C	Ассемблер	

void main()
{
// вызов
int k=4;
Increment(k);
// здесь k=5
}
 	
void Increment(int *kf)
{
kf=kf+1;
}
	Increment proc far
	push BP
	mov BP, SP
	//стек: BP,IP,CS, смещ. и сегмент перем.
	push BX
	push DS
	lds BX, SS:[BP+6] //загр. указателя
	// DS:BX - адрес переменной
	inc word ptr DS:[BX]
	pop DS
	pop BX
	pop BP
	ret 4 //удалить смещ. и сегмент перем.
	Increment endp
	
	// вызов (k объявлена в сегменте DS)
	lea AX, k
	push DS
	push AX
	call Increment
	// здесь k=5
		Increment proc
		inc word ptr DS:[BX]
		ret
		Increment endp
		
		// вызов
		lea BX, k
		call Increment
		// здесь k=5

2.2) условной передачи:

- jxx <имя метки> – передает управление на метку если выполняется условие xx, задаваемое в команде: числа со знаком g – (greater) больше, l – (less) меньше; числа без знака a – (above) больше, b – (below) меньше; e – (equalization) равно; c, z, p, s, o – переход если соответствующий флаг установлен; n – применяется в сочетании с другими условиями для их отрицания;
- loop <имя метки> – завершение цикла, команда декрементирует регистр CX и передает управление на метку до тех пора пока регистр CX отличен от нуля (цикл с постусловием), дополнительно команда допускает суффиксы z (выполнять пока ноль), e (выполнять пока равно) и их отрицание (nz, ne соответственно).
  Пример
  mov CX, 14
  metka: //произвольное сочетание букв, цифр, подчеркивания, с последующим символом «:»
  <тело цикла – набор команд, которые повторятся CX раз>
  loop metka //конец цикла, выполняется: CX=CX-1 и переход на metka если CX>0

3. **Арифметико-логические команды**:

3.1) арифметические команды над двоичными операндами

- add/sub –сложение/вычитание (операнды: RG,RG; RG,mem; mem,RG; RG,const; mem,const);
- adc/sbb – сложение/вычитание с учетом переноса/заема (операнды: см. add/sub);
- inc/dec – инкремент/декремент на 1 указанного операнда: RG или mem;
- mul *src* (imul *src*) – беззнаковое (знаковое) умножение (операнд *src* это RG или память): dx:ax=ax**src* если *src* – 16 разрядный операнд и ax=al**src* если *src* – 8 разрядный;
- div *src* (idiv *src*) – беззнаковое (знаковое) деление: ax=dx:ax/*src*, dx содержит остаток если *src* –16 разрядный операнд и al=ax/*src*, ah содержит остаток если *src* – 8 разрядный.
- neg – изменение знака операнда (RG или mem)
- cmp – арифметическое сравнение (производится вычитание второго аргумента из первого, результат не сохраняется, по результату выставляются флаги), операнды: RG,RG; RG,mem; mem,RG; RG,const; mem,const
- cmpsb (cmpsw) – арифметическое сравнение байтов (слов) с адресами ds:si и es:di, флаги по результату, с инкрементом или декрементом si и di на 1 (2) в соответствии с флагом направления (DF);

3.2) логические команды (выполняются поразрядно):

- and – конъюнкция, or – дизъюнкция, xor – исключающее или (неравнозначность) операнды: RG,RG; RG,mem; mem,RG; RG,const; mem,const
- not – побитная инверсия (операнд: RG или mem)
- shl/shr (sar) – логический сдвиг влево/вправо (арифметический сдвиг вправо) на 1 разряд, заполнение нулями (знаковым разрядом), а выдвигаемый бит копируется в флаг carry (операнд: RG или mem)
- rol/ror – циклический сдвиг влево/вправо на 1 разряд, выдвигаемый бит копируется в флаг carry (операнд: RG или mem)
- rcl/rcr – циклический сдвиг влево/вправо на 1 разряд при этом carry флаг интерпретируется как дополнительный бит регистра (операнд: RG или mem)
- test – логическое сравнение (производится логическое умножение операндов, результат не сохраняется, по результату выставляются флаги), операнды: RG,RG; RG,mem; RG,const; mem,const
- scasb (scasw) – логическое сравнение AL (AX) с байтом (словом) по адресу ES:[DI] с инкрементом или декрементом di на 1 (2) в соответствии с флагом направления (DF)

(*) команды сдвига допускают наличие второго операнда – регистра CL, указывающего на сколько разрядов сдвигается операнд.

3.3) команды двоично-десятичной арифметики (в старших и младших 4 битах (тетрадах) содержится число от 0 до 9, т.е. байт в двоично-десятичной интерпретации представляет собой число от 0 до 99 – упакованное двоично-десятичное число, если используется только младшие 4 бита – неупакованное двоично-десятичное число):

- DAA (коррекция после сложения упакованных чисел) выполняет следующие действия (используется после команд add, adc, inc):
  1) Если младшие 4 бита AL больше 9 или флаг AF=1, то AL увеличивается на 6; флаги CF и AF устанавливаются, если при этом сложении произошёл перенос, иначе AF=0
  2) Если теперь старшие 4 бита AL больше 9 или CF=1, то AL увеличивается на 60h и CF=1, иначе CF=0
  3) флаги ZF, SF и PF устанавливаются в соответствии с результатом, флаг OF не определён
  Пример: AL=19h, после последовательности команд: inc AL и daa в регистре AL находится число 20h (а не 1Ah).
- DAS (коррекция после вычитания упакованных чисел) выполняет следующие действия (используется после команд sub, sbb, dec):
  1) Если младшие 4 бита AL больше 9 или AF=1, то AL уменьшается на 6; флаги CF и AF устанавливаются, если при этом вычитании произошёл заём, иначе AF=0
  2) Если теперь старшие 4 бита AL больше 9 или CF=1, то AL уменьшается на 60h и CF=1, иначе CF=0
  3) флаги ZF, SF и PF устанавливаются в соответствии с результатом, флаг OF не определён
  Пример: AL=20h, после последовательности команд: dec AL; das в регистре AL находится число 19h (а не 1Fh).
- AAA – корректирует сумму двух неупакованных двоично-десятичных чисел в AL. Если коррекция приводит к десятичному переносу, то AH увеличивается на 1. Флаги AF=CF=1, если произошёл перенос между тетрадами AL, иначе AF=CF=0. Значения флагов ZF, SF, OF, PF не определены.
  Пример: при сложении 05 и 06 в AX окажется число 000Bh, то последующая команда aaa скорректирует его в 0101.
- AAS – корректирует разность двух неупакованных двоично-десятичных чисел в AL. Если операция приводит к займу, то AH уменьшается на 1. Флаги AF=CF=1, если произошёл заём между тетрадами AL, иначе AF=CF=0. Значения ZF, SF, PF, OF не определены.
- AAM – корректирует результат умножения неупакованных двоично-десятичных чисел, который находится в AX после выполнения команды mul. Флаги ZF, SF, PF устанавливаются в соответствии с результатом, CF, OF, AF не определены.
  Пример: mov AL,5; mov BL,5; mul BL; aam. После этого AX=0205h, а не 0019h.
- AAD – корректирует неупакованное двоично-десятичное число в AX перед делением. Флаги ZF, SF, PF устанавливаются в соответствии с результатом, CF, OF, AF не определены.
  Пример: mov AX,0205h; mov BL,5; aad; div BL. После этого AX=0005.

4. **Команды управления**
   - команды сброса/установки битов регистра флагов: cli/sti (флаг прерываний); cld/std (флаг направления); clc/stc (флаг переноса-заема); cmc – инверсия флага переноса;
   - остановка до прерывания: halt;
   - ожидание активного сигнала на линии *test* МП: wait
5. **Прочие команды**

5.1) холостая команда (no operation): nop

5.2) префиксы:
   - rep – повторить команду CX раз;
   - repz (repnz) – повторить команду CX раз если флаг нуля сброшен (установлен);
   - repe (repne) – повторить команду CX раз если равно (неравно);
   - lock – запрет захвата шины на время выполнения команды

Примеры программ

1) Вычислить сумму элементов массива

```
.model tiny // определяет модель программы: tiny - код, данные и стек в одном сегменте
.code //сегмент кода
org 100h //смещение под заголовок исполняемого файла - тип COM
start:
```

```
mov CX, 5       //CX=число элементов массива mas
xor AX, AX      //AX=0: AL=0 и AH=0
mov sum, AX  //инициализация переменной
lea SI, mas      //DS:SI – адрес первого байта массива
cld
L01:
  lodsb          //AL=DS:[SI], SI=SI+1, AH –не изменяется (инициализирован AH=0)
  add sum, AX
  loop L01
// sum – содержит сумму элементов массива
ret
//сегмент данных – совпадает с сегментом кода
mas db 0,1,2,3,4 // байтовый массив
sum dw 0 // переменная (16 бит)
end start          //точка входа в программу
```

2) Найти максимальный элемент в массиве

```
mov CX, 7       //CX=число элементов массива
xor AX, AX
lea DI, mas
push DS
pop ES
cld
L01:
  scasb          //сравнение AL и ES:[DI], DI=DI+1
  ja L02         // пропустим следующую команду, если AL>элемент массива ES:[DI]
    mov AL, ES:[DI-1]          // копируем большее число в AL из массива
  L02:
  loop L01
mov max, AL   // max - максимальный элемент массива
ret
mas db 8,10,20,0,7,4,10 // байтовый массив с данными
max db 0 // сюда запишем результат работы программы
```

3) Вычислить скалярное произведение двух векторов (с неотрицательными координатами)

```
mov CX, 4       //CX=длина векторов
mov si, offset x1
mov di, offset x2
xor AX, AX
xor BX, BX
mov result, AX          mov result_h, AX
cld
L01:
  lodsb          //AL=DS:[SI] – координата вектора x1
  mul byte ptr DS:[DI]          //AX=AL* DS:[DI] – произведение координат векторов
  inc DI         //инкремент адреса элемента массива x2
  add result, AX
  adc result_h, BL  // учет возможного переноса за пределы 16-разрядного числа
  loop L01
// result – содержит скалярное произведение векторов x1 и x2
ret
x1 db 8,10,20,0 //координаты вектора в 4-х мерном пространстве
x2 db 2,1,4,10
result dw 0       //результат вычисления требует 18 разрядов
result_h db 0
```

3. Периферия процессора ВМ86

Под периферией процессора понимается набор электронных устройств, которые обеспечивают (совместно с процессором) функционирование внешних устройств хранения, обработки и передачи данных – рис.14. Такой набор устройств обычно включает в себя: память (ОЗУ, ПЗУ), контроллеры прерываний и прямого доступа к памяти, контроллер внешних портов (последовательный и параллельный), микросхему часов реального времени, контроллер накопителя на магнитных (оптических и т.п.) дисках, контроллер (процессор) отображения графической и текстовой информации, сетевой контроллер и т.п.

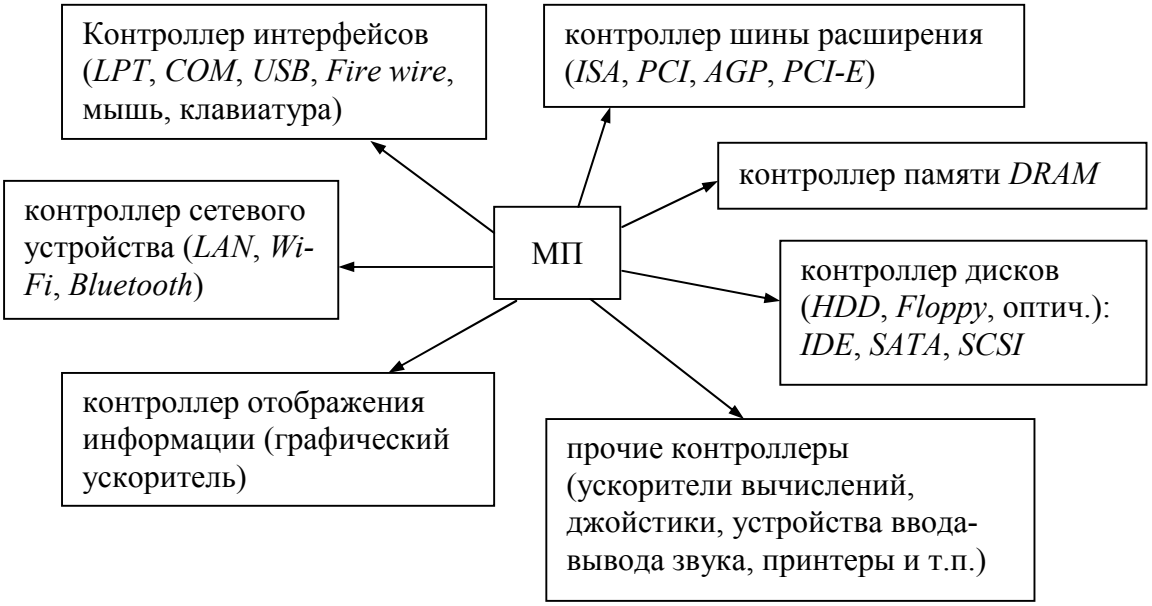


Рис. 14 – "Окружение" микропроцессоров x86

В первых компьютерах эти контроллеры реализовывались набором внешних микросхем, позже их стали объединять в рамках одной, двух СБИС – их называли *chipset*. Некоторые, такие как графические ускорители, контроллер внешних магнитных, оптических и электронных (твердотельных) дисков, выделились (обособились) и со временем по важности встали на один уровень с центральным процессором.

3.1 ПОДКЛЮЧЕНИЕ ПАМЯТИ

Для функционирования процессора необходимо подключение памяти, в которой хранится программа и исходные, промежуточные, выходные данные. Обычно всю память делят на память программ и память данных. Однако в системах на универсальных процессорах память программ и данных представляет собой единое пространство адресов и физически реализуется в одном или нескольких банках памяти. К примеру, в микроконтроллерах напротив эти памяти четко разделяют: разные физические шины и разные банки памяти.

Деление пространства адресов памяти МП на память программ и данных обычно осуществляется программно-аппаратным путем, например: считают что 20-разрядный адрес с установленным старшим битом – это адресное пространство ПЗУ, а со сброшенным старшим битом адреса – ОЗУ (можно разделить по двум старшим битам, регулируя таким образом соотношение объемов ПЗУ и ОЗУ в общем пространстве адресов). В более общем случае разделение памяти на ПЗУ и ОЗУ можно осуществить дешифрацией старших разрядов адреса. При этом требования для системы на базе ВМ86 состоят в непрерывности области адресов и в обязательном расположение ПЗУ с адреса FFFF0h (с которого x86 начинает выполнять инструкции – см. раздел сигнал *Reset* МП), также по адресам 00000-003FFh (первый килобайт) должно размещается ОЗУ, где хранится таблица векторов прерываний (см. раздел о системе прерываний МП и соответствующем контроллере).

В общем случае при подключении ЗУ к шинам МП необходимо обеспечивать передачу как двухбайтовых слов, так и отдельных байтов. С этой целью память выполняется в виде двух банков (в качестве примера ОЗУ и ПЗУ по 8 кбайт – рис.15): младшего, подключаемого к линиям данных *D*7-*D*0 и содержащего байты с четными адресами (*A*0=0), и старшего, соединенного с *D*15-*D*8 и содержащего байты с нечетными адресами (*A*0=1).

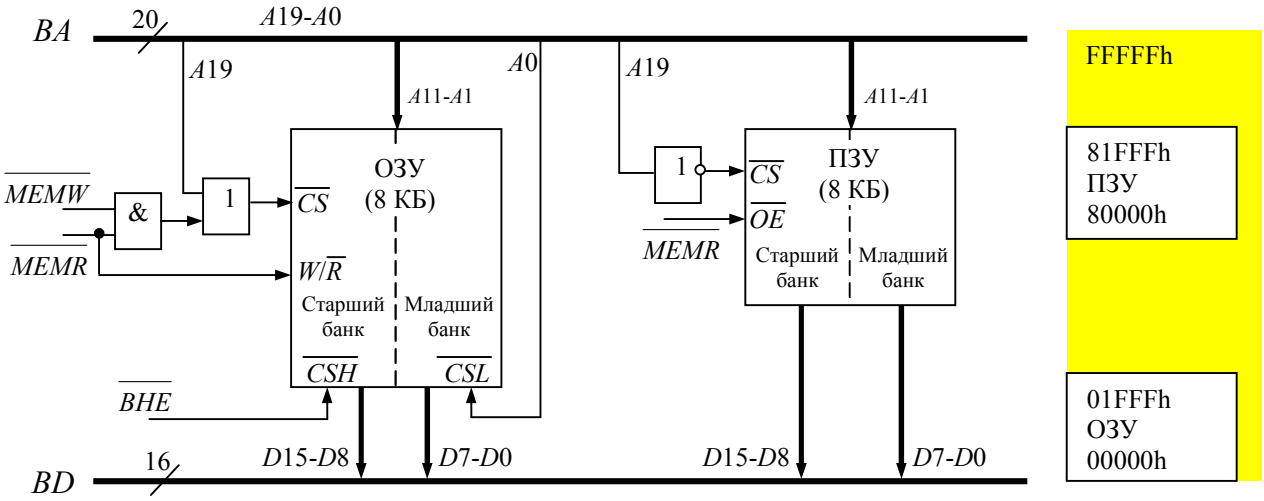


Рис. 15 – Схема подключения банков памяти и карта памяти (справа)

Чтобы каждое слово передавалось за один цикл шины, слова располагают только с четных адресов. Напомним, что адресная линия *A*0 совместно с линией разрешения старшего банка *BHE* обеспечивает следующие варианты пересылок по шине данных:

*A*0=0, *BHE*=0 – пересылается слово;
*A*0=0, *BHE*=1 – пересылается только младший байт;
*A*0=1, *BHE*=0 – пересылается только старший байт;
*A*0=1, *BHE*=1 – устройство не выбрано.

Выработка сигнала *BHE* и указанный порядок пересылок реализуются МП автоматически. При записи в ОЗУ также необходимо различать старший и младший байты, что обеспечивается подачей сигналов *BHE* и *A*0 на входы *CSH* и *CSL* выбора старшего и младшего банков ОЗУ.

При чтении из ПЗУ в любом случае на шину данных будет подаваться слово, из которого МП при необходимости выберет требуемый байт и поместит его в регистр, указанный в выполняемой команде. Поэтому сигналы *BHE* и *A*0 на ПЗУ не подаются.

3.2 СИСТЕМА ПРЕРЫВАНИЙ ПРОЦЕССОРА

Прерывание – это аппаратная реализация отклика синхронной системы на асинхронное событие. В ВМ86 реализована многоуровневая система прерываний по вектору с числом векторов до 256. Адреса подпрограмм прерывания занимают область ОЗУ емкостью 1 Кбайт, которая располагается в младших адресах памяти: 00000-003FFh (этот участок памяти называют таблицей векторов прерываний). Таким образом, каждый адрес вектора прерываний занимает 4 байта в форме *seg:ofs* программы обработчика прерывания (является указателем на процедуру). Задачей прерывания является приостановка выполнения текущей программы, передача управления и выполнение программы обработчика (процедуры) поступившего прерывания, возврат управления и продолжение выполнения прерванной задачи. Все прерывания разделены на две группы: аппаратные и программные. Последние инициируются специальной командой (*INT*) непосредственно в программе.

Аппаратные прерывания предназначены для немедленной реакции процессора на какое-либо асинхронное к нему событие (например: клавиатура, системный таймер, мышь, модем, сетевой интерфейс и т.п.). Для организации прерываний процессор имеет специальные внешние выводы:

- NMI (*Non Mascable Interrupt*) – немаскируемое прерывание, распознается МП по завершению текущей команды независимо от состояния флага разрешения прерывания *IF*. Высокий уровень на этом входе вызовет прерывание с номером вектора 2, которое выполняется так же, как и маскируемое. Его обработка не может прерываться под действием сигнала на входе *NMI* до выполнения команды *IRET*. Этот вход предназначен для сигнализации о некоторых критических ситуациях, например: об аварийном отключении сетевого питания.
- INTR – запрос прерывания (маскируемый), опрашивается центральным процессором в конце выполнения каждой команды, если прерывания разрешены (*IF*=1), то запрос фиксируется во внутреннем триггере. Обычно на вход *INTR* подается запрос от программируемого контроллера прерываний К1810ВН59А. Если *IF*=0, то запрос по входу *INTR* игнорируется.
- INTA – подтверждение запроса прерывания, формируется в ответ на принятый запрос прерывания *INTR*, выполняет функцию сигнала *RD* в цикле подтверждения прерывания и стробирует считывание указателя адреса (номера вектора) прерывания. В каждом случае подтверждения прерывания выполняются два цикла *INTA*, первый из которых является предварительным и не сопровождается чтением информации.

Самостоятельно процессор в состоянии обслужить только одно маскируемое и одно немаскируемое прерывание. Для увеличения числа маскируемых прерываний применяется контроллер прерываний i8259*A* (аналог К1810ВН59А), который подключается к выводам *INTA* и *INTR* МП. Для дальнейшего увеличения числа прерываний применяют каскадное включение контроллеров, один из которых будет ведущим (*master*), а другие ведомыми (*slave*).

Рассмотрим как МП подтверждает прерывание после восприятия запроса по входу *INTR* – рис.16. Запрос по входу *INTR* фиксируется внутри МП триггером (длительность импульса запроса должна быть не менее периода тактирующих импульсов к моменту когда МП закончит текущий цикл). МП выполняет два так называемых цикла *INTA*, разделенные двумя холостыми тактами *TW*. В этих циклах МП не выдает адрес, но формирует строб *ALE*. Первый цикл обеспечивает подготовку к приему байта, определяющего тип прерывания (номер вектора в таблице прерываний), и информация в этом цикле не принимается. Во втором цикле по линиям *AD*7-*AD*0 МП читает номер вектора прерывания (по которому в таблице прерываний определяется адрес процедуры обработки прерывания). Этот цикл подобен обычному циклу чтения, но вместо сигнала *RD*=0 вырабатывается сигнал *INTA*=0. Для предотвращения захвата шин сигналом на входе *HOLD* (или *RQ*/*GT* в максимальном режиме) формируется внутренний сигнал блокировки *LOCK*, начиная с такта *T*2 первого цикла и кончая тактом *T*2 второго цикла сигнала *INTA*.

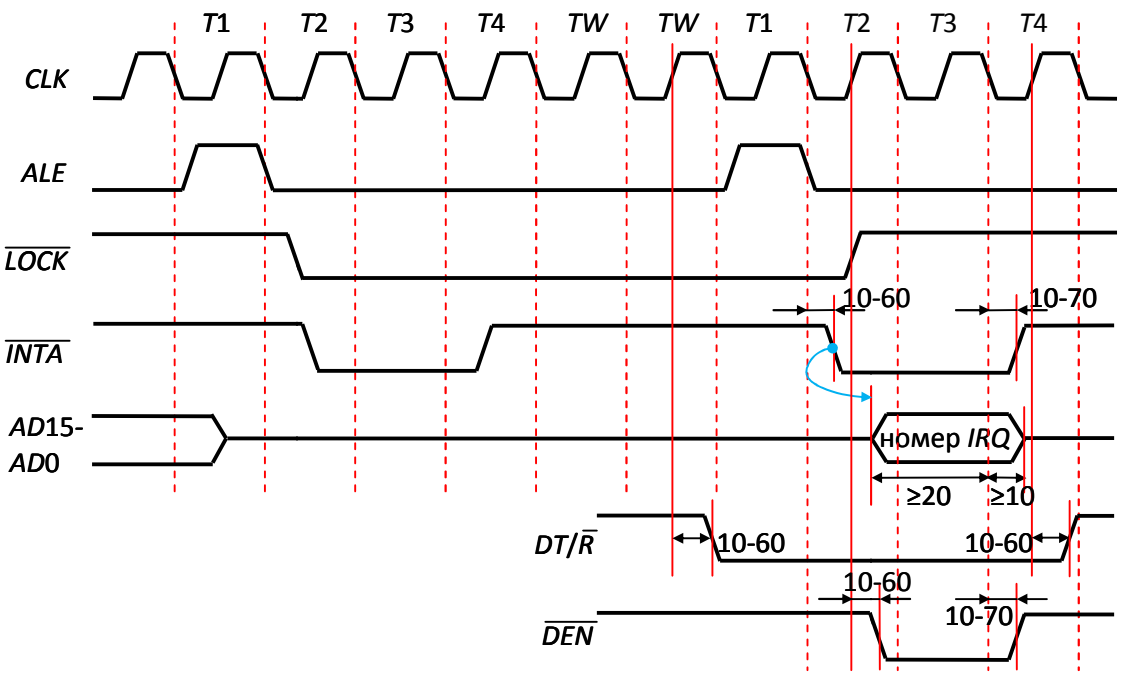


Рис. 16 – Последовательность сигналов при подтверждении прерывания по входу *INTR* (минимальный режим)

В максимальном режиме i8086 работает совместно с контроллером шины, поэтому цикл подтверждения "выглядит иначе" – рис.16а (роль строба *ALE* выполняет сигнал контроллера шины *MCE*/*PDEN*, активный уровень сигнала *DEN* – высокий, в первом такте на шине *AD* резервируется интервал для передачи адресной информации *CAS* - *cascade addr*).

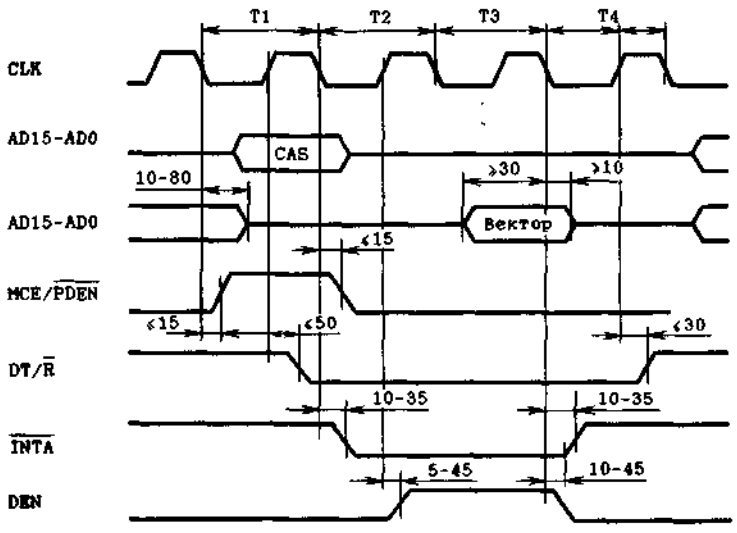


Рис. 16а – Циклы подтверждения прерывания в максимальной режиме (второй цикл *INTA*)

3.2.1 КОНТРОЛЛЕР ПРЕРЫВАНИЙ

Задача программируемого контроллера прерываний (ПКП) состоит:

- в приеме запросов прерывания от ВУ;
- в сравнении их приоритетов и посылки запроса прерывания в ЦП вместе с информацией о номере запроса (по которому можно вызвать процедуру обработки этого запроса).

Микросхема К1810ВН59А (i8259A) выпускается в 28-контактном корпусе типа *DIP* и требует один источник питания напряжением +5 В. На рис.17 представлена структурная схема контроллера.

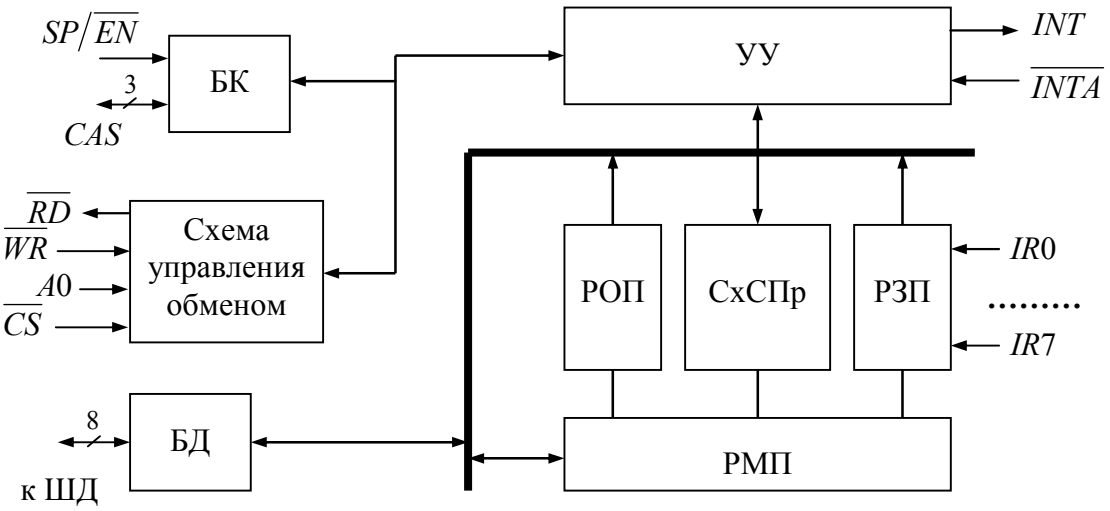


Рис. 17 – Схема программируемого контроллера прерываний ВН59А

Буфер данных (БД) вместе со схемой управления обменом по ШД *D*7-*D*0 обеспечивает прием управляющих слов (команд) в контроллер при программировании, а также выдачу состояний регистров контроллера и номера вектора прерываний.

Регистр запросов прерываний (РЗП) служит для запоминания всех запросов от ВУ по входам *IR*7-*IR*0.

Регистр масок прерываний (РМП) хранит маску, с помощью которой можно запретить обслуживание запросов по любому отдельно взятому входу или их группе.

Схема сравнения приоритетов (СхСПр) выбирает запрос с наибольшим приоритетом среди вновь поступивших и уже обслуживаемых запросов.

Регистр обслуживаемых прерываний (РОП) содержит единицы на позициях, соответствующих обслуживаемым запросам, причем каждая единица запрещает обслуживание запросов с меньшим приоритетом (кроме режима спецмаскирования).

Управляющее устройство (УУ), содержащее регистры команд инициализации РКИ (*ICW*) и регистры рабочих команд РРК (*OCW*), обеспечивает выработку внешних и внутренних управляющих сигналов.

Блок каскадирования (БК) осуществляет связь ведущей БИС контроллера с ведомыми при использовании нескольких ПКП в микропроцессорной системе.

Для понимания работы ПКП с ВМ86 рассмотрим последовательность его действий:

1. ПКП ВН59А воспринимает запросы прерывания по входам *IR* от одного или нескольких ВУ, записывает единицы в соответствующие разряды РЗП, проверяет маскирование, определяет среди незамаскированных (разрешенных) прерываний запрос с наивысшим приоритетом и посылает сигнал прерывания *INT* на вход *INTR* МП.
2. Если в МП прерывания разрешены (флаг *IF*=1), то МП заканчивает выполнение текущей команды и активизирует сигнал подтверждения прерывания *INTA* (рис. 16 – первый цикл *INTA*).
3. По получению первого импульса *INTA* ПКП устанавливает в «1» разряд регистра РОП, соответствующий запросу, подлежащему обслуживанию, а одноименный разряд регистра РЗП сбрасывается в «0».

4. МП вырабатывает второй импульс *INTA*, по получению которого контроллер посылает по ШД в МП 8-разрядный указатель (вектор), используемый для определения начального адреса подпрограммы обслуживания прерывания (по таблице адресов прерываний).
5. МП, сохраняя в стеке регистр флагов и адрес точки возврата (т.е. текущие значения регистров *CS*:*IP*), переходит на выполнение подпрограммы обслуживания прерывания (по окончанию которой командой *IRET* будут извлечены из стека значение регистра флагов и адрес команды прерванной программы, на которую будет передано управление).

Контроллер программируется с помощью команд, которые формируются в регистре *AL* центрального процессора и передаются в ПКП по команде *OUT*.

Различают два вида команд программирования:

- команды инициализации *ICW*, используемые для начальной подготовки ПКП к работе;
- рабочие команды *OCW*, предназначенные для задания маски, различных режимов работы ПКП, а также для обеспечения контрольного считывания содержимого регистров ПКП.

Управляющие сигналы, которые сопровождают подачу команд, приведены в табл.7.

Таблица 7

*A*0	*D*4	*D*3	*RD*	*WR*	*CS*	Операция
0	0	0	1	0	0	ШД → *OCW*2 (ПКП получает рабочую команду)
0	0	1	1	0	0	ШД → *OCW*3
0	1	X	1	0	0	ШД → *ICW*1 (ПКП получает команду инициализ.)
1	X	X	1	0	0	ШД → *ICW*2, *ICW*3, *ICW*4, *OCW*1
0	–	–	0	1	0	РЗП, РОП → ШД (ПКП выдает значения *RG*)
1	–	–	0	1	0	РМП → ШД
X	X	X	1	1	0	Нет операции (выходы *D*7-*D*0 в третьем состоянии)
X	X	X	X	X	1	

(*) в адресном пространстве портов каждый ПКП требует выделения двух портов

Последовательность команд инициализации (рис.18) состоит из трех команд *ICW*1, *ICW*2, *ICW*4, если используется один ПКП в системе, либо из четырех команд *ICW*1-*ICW*4.

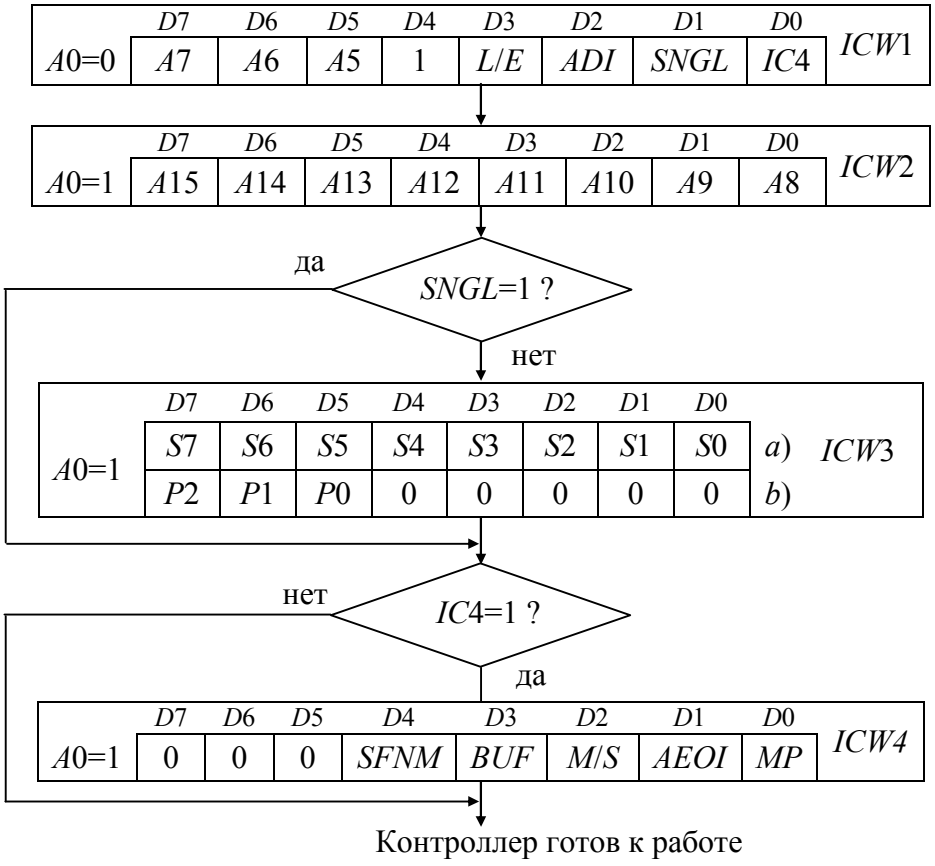


Рис. 18 – Последовательность команд начальной инициализации ПКП

Рассмотрим назначение полей каждой из команд.

Команда ICW1:

- *IC*4 (бит 0) – указывает на наличие команды *ICW*4 в последовательности инициализации (при работе с ВМ86 всегда *IC*4=1);
- *SNGL* (бит 1) – указывает на использование одного контроллера прерываний (если *SNGL*=0, то контроллеров несколько, в системах на ВМ86 их два);
- *ADI* (бит 2) – в системах с ВМ86 не используется (*ADI* =0);
- *L/E* (бит 3) – определяет способ воздействия сигналов запросов прерывания на входы *IR*: при *L*/*E*=1 входы *IR* являются потенциальными, при *L*/*E* =0 – импульсными, т.е. запись единицы в регистр РЗП происходит по фронту импульса запроса, для чего в контроллере подключается логическая схема анализа фронтов (следует учитывать, что запрос на прерывание должен быть снят до поступления команды окончание прерывания *EOI* или перед разрешением прерывания в МП, т.е. перед установкой *IF*=1, чтобы предотвратить повторное прерывание от того же запроса; в этом смысле использование запроса по фронту предпочтительнее, так как при этом не накладываются ограничения на длительность сигнала запроса);
- *D*4 (бит 4) =1 – указывает в совокупности с *A*0=0, что подается команда *ICW*1;
- *A*7-*A*5 (биты 5-7) – в системах на ВМ86 эти разряды игнорируются.

Команда ICW2

Команда указывает значение разрядов адреса *A*15-*A*8 – интерпретируются как смещение в таблице прерываний, т.е. задают начальный номер *IRQ*, относительно которого размещаются запросы этого ПКП (в системе на ВМ86 принимаются во внимание лишь разряды *A*15-*A*11, а разряды *A*10-*A*8 принимаются равными нулю).

Команда ICW3

Команда указывает места подключения ведомых контроллеров к ведущему:

a) для ведущего контроллера S_i=1 означает, что к его входу *IRi* подключен ведомый контроллер;

b) для ведомого контроллера двоичный код $P_2P_1P_0$ указывает номер входа IR_i ведущего контроллера, к которому подключен данный ведомый контроллер.

Команда ICW4

Команда несет следующую информацию:

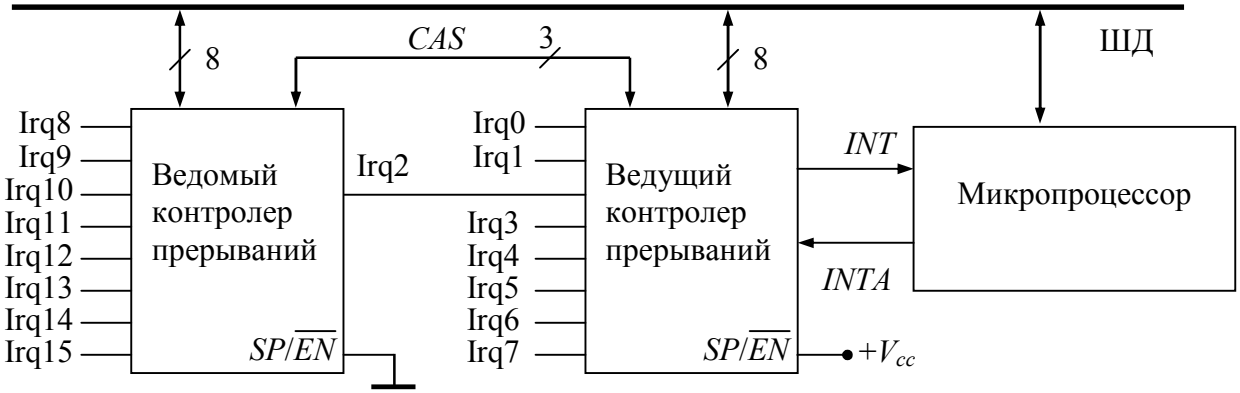
- *MP* указывает тип микропроцессора, с которым взаимодействует контроллер: *MP*=1 соответствует ВМ86, *MP*=0 –ВМ80;
- *AEOI* указывает режим окончания прерывания (*EOI*): *AEOI*=1 – задает автоматическое окончание, когда соответствующий разряд регистра РОП сбрасывается в начале программы обслуживания текущего прерывания, *AEOI*=0 – задает режим обычного окончания, при котором соответствующий разряд РОП сбрасывается программистом по окончанию программы обслуживания;
- *BUF* означает, что контроллер запрограммирован для работы с буферизацией ШД (такая буферизация обязательна, если к ШД подключено большое число устройств); если *BUF*=1, то выход *SP*/*EN* становится выходом управляющего сигнала для буфера, причем ведущий контроллер отличается от ведомого значением разряда *M*/*S*=1 и *M*/*S*=0 соответственно; если *BUF*=0, то разряд *M*/*S* игнорируется;
- *SFNM* указывает на то, что контроллер запрограммирован на специальный режим полной вложенности, который будет пояснен ниже.

В результате инициализации по умолчанию контроллер оказывается готовым к работе в режиме полной вложенности:

- регистр маски РМП сброшен;
- входу *IR*7 присвоен низший приоритет 7;
- триггер режима спецмаскирования сброшен;
- указатель источника информации при чтении статуса установлен на РЗП;
- если *IC*4=0, то все функции, устанавливаемые командой *ICW*4, соответствуют нулевым

значениям разрядов этой команды.

Типовая схема включения ПКП в системах на основе i8086 представлена на рис.19.



Порты доступа: A0h, A1h

Порты доступа: 20h, 21h

Программирование ведомого

ICW1	0001 0001	
ICW2	0111 0000	базовый адрес 70h
ICW3	0100 0000	подключен к Irq2
ICW4	0000 0001	

Программирование ведущего

ICW1	0001 0001	
ICW2	0000 1000	базовый адрес 08h
ICW3	0000 0100	к Irq2 включен slave
ICW4	0000 0001	

Рис. 19 – Типовая схема включения ПКП в системе на основе *i*8086

Типовое распределение аппаратных прерываний в *PC* на основе *x*86:

Уровень прерывания	Вектор прерывания	Устройство по умолчанию
IRQ0	08h	Таймер
IRQ1	09h	Клавиатура
IRQ2	0Ah	Вход ведомого
IRQ8	70h	КМОП-микросхема
IRQ9	71h	Перенаправлено на int 0Ah
IRQ10	72h	Зарезервировано
IRQ11	73h	Зарезервировано
IRQ12	74h	Мышь (PS/2)
IRQ13	75h	Исключение сопроцессора
IRQ14	76h	Жесткий диск
IRQ15	77h	Зарезервировано
IRQ3	0Bh	Последовательный порт COM2
IRQ4	0Ch	Последовательный порт COM1
IRQ5	0Dh	Порт LPT2
IRQ6	0Eh	Гибкий диск
IRQ7	0Fh	Порт LPT1

3.2.2 ОСНОВНЫЕ РЕЖИМЫ И КОМАНДЫ КОНТРОЛЛЕРА

Различают четыре основные режима работы контроллера:

- полной вложенности,
- циклического приоритета,
- спецмаскирования,
- поллинга (последовательного опроса).

Режим полной вложенности, при котором каждый вход запроса прерываний имеет фиксированный приоритет от высшего уровня (вход *IR*0) до низшего (вход *IR*7), является стандартным и не требует использования рабочих команд *OCW*. Остальные режимы, а также

состояние регистра маски РМП могут быть заданы с помощью следующих рабочих команд (табл.8).

Таблица 8

Команда	*A*0	*D*7	*D*6	*D*5	*D*4	*D*3	*D*2	*D*1	*D*0
*OCW*1	1	*M*7	*M*6	*M*5	*M*4	*M*3	*M*2	*M*1	*M*0
*OCW*2	0	*R*	*SEOI*	*EOI*	0	0	*L*2	*L*1	*L*0
*OCW*3	0	0	*SSM*	*SM*	0	1	*P*	*SRTS*	*RIS*

Команда OCW1

Команда *OCW*1 устанавливает и сбрасывает разряды регистра РМП: M_i=1 означает, что в *i*-й разряд РМП записывается единица, которая маскирует вход IR_i, во всех режимах, в результате чего запросы по этому входу будут игнорироваться.

Команда OCW2

Команда *OCW*2 управляет режимами сдвигов приоритетов и окончаний прерываний путем установки разрядов *R*, *SEOI* и *EOI* в соответствии с табл.9.

Поле *L*2-*L*0 определяет двоичный код уровня прерывания, который получает низший приоритет, когда *SEOI*=1.

Таблица 9

Разряды *OCW*2			Выполняемые действия
R	*SEOI*	*EOI*	
0	0	0	Сброс триггера циклического сдвига режима *A*
0	0	1	Неспецифическое окончание *EOI*
0	1	0	Нет операции
0	1	1	Специфическое окончание *EOI*
1	0	0	Установка триггера циклического сдвига режима *A*
1	0	1	Автоматический циклический сдвиг при *EOI* (режим *A*)
1	1	0	Циклический сдвиг приоритетов (режим *B*) независимо от *EOI*
1	1	1	Циклический сдвиг приоритетов при *EOI* (режим *B*)

Команда OCW3

Команда *OCW*3 служит для установки режимов маскирования, считывания статуса и поллинга:

- Разряд *SSM* разрешает разряду *SM* управлять режимом спецмаскирования (когда *SSM*=0, разряд *SM* не действует):
  - при *SM*=1 – контроллер перейдет к режиму спецмаскирования;
  - при *SM*=0 – вернется к нормальному режиму маскирования.
- Разряд *ERIS* разрешает считывание состояния регистров контроллера, разряд *RIS* осуществляет выбор регистра:
  - при *RIS*=1 считывается РОП (по сигналу *RD*);
  - при *RIS*=0 – РЗП.
- Разряд *P* позволяет установить контроллер в режим поллинга (последовательного опроса).

Как уже отмечалось, начало обслуживания прерывания фиксируется путем установки в «1» соответствующего разряда регистра РОП. Сброс этого разряда осуществляется автоматически по нарастающему фронту последнего импульса *INTA* (при *AEOI*=1 в команде *ICW*4) либо командой *EOI*, которая должна быть включена в подпрограмму обработки прерывания. В случае каскадирования контроллеров команда *EOI* должна выдаваться дважды: один раз для ведущего и один раз для ведомого контроллера.

Различают две формы команды *EOI*: специфическую и неспецифическую. Когда контроллер работает в режиме полной вложенности, он может самостоятельно определить номер разряда РОП, который нужно сбросить по команде *EOI*. В этом случае подается неспецифическая команда *EOI*, и контроллер сбросит младший разряд в РОП из тех, которые содержат единицы, так как в режиме полной вложенности обслуживаемый уровень с меньшим

номером всегда является наивысшим. При использовании режима, который может изменить полную вложенность, контроллер не в состоянии определить последний обслуживаемый уровень, чтобы сбросить соответствующий разряд в РОП. В этом случае подается специфическая команда *SEOI* с помощью *OCW*2, в которой *SEOI*=1, а разряды *L*2–*L*0 указывают номер разряда РОП, подлежащий сбросу.

Команда *EOI*, выполняемая по заднему фронту последнего импульса *INTA*, в режиме автоматического окончания прерывания является неспецифической. Отметим, что с системной точки зрения этот режим может использоваться только тогда, когда в пределах одного подчиненного контроллера не требуется обеспечивать полную вложенность прерываний.

3.2.3 РЕЖИМЫ РАБОТЫ КОНТРОЛЛЕРА

Режим полной вложенности.

Как было описано выше, этот режим устанавливается сразу после окончания инициализации. Приоритеты запросов прерывания упорядочиваются в сторону убывания от 0 до 7. Добавим, что в системе с каскадированием контроллеров полная вложенность обеспечивается только по входам ведущего контроллера. При обслуживании запроса по какому-либо входу ведомого контроллера запросы по другим его входам (даже с более высоким приоритетом) не обслуживаются.

Специальный режим полной вложенности.

Применяется тогда, когда в системе используется каскадирование контроллеров и вложенность приоритетов должна сохраняться для каждого контроллера. Этот режим, устанавливаемый командой *ICW*4, отличается от обычного режима полной вложенности следующим:

1) Когда обслуживается запрос от некоторого ведомого контроллера, этот контроллер не закрыт от приоритетной логики ведущего контроллера. Дальнейшие запросы на прерывание от входов с более высоким приоритетом в пределах ведомого контроллера будут распознаваться ведущим контроллером и инициировать запрос прерывания для МП (в обычном режиме полной вложенности ведомый контроллер маскируется, если его запрос находится в обслуживании, так что другие запросы от него не воспринимаются).
2) Если запускается подпрограмма обслуживания прерывания, то программным путем необходимо проверить является ли данное прерывание единственным от этого ведомого контроллера. Это делается с помощью посылки команды неспецифического окончания прерывания (*EOI*) на подчиненный контроллер с последующим чтением содержимого РОП и проверкой его на нуль. Если проверка удовлетворена, то такая же команда *EOI* посылается на ведущий контроллер для анализа его РОП.

Режим циклического приоритета.

Используется в тех случаях, когда требуется установить одинаковые приоритеты на обслуживание *IRQ*. При этом любое обслуженное *IRQ* получает низший приоритет, а приоритеты остальных устройств циклически сдвигаются относительно него. Повторное обслуживание *IRQ* возможно лишь после обслуживания всех остальных устройств. Такой режим автоматического сдвига приоритетов называется режимом *A* и устанавливается командой *OCW*2 при *R*=1, *EOI*=1 и *SEOI*=0. Новое распределение приоритетов фиксируется по команде *EOI*. Если *R*=1, *EOI*=*SEOI*=0, то по команде *OCW*2 устанавливается в «1» специальный триггер «вращать –*A*», находящийся в УУ, и тогда перераспределение приоритетов осуществляется по команде *AEOI*, описанной выше.

В контроллере имеется также режим программного сдвига приоритетов – режим *B*. В этом случае программист может с помощью команды *OCW*2 при *R*=*SEOI*=1 задавать номер входа *L*2-*L*0, которому будет присвоен низший приоритет, а приоритеты других входов зафиксируются по отношению к заданному. Так, если *IR*5 будет запрограммирован в качестве входа с низшим приоритетом, то *IR*6 получит самый высокий приоритет, причем изменение внутреннего статуса приоритетов произойдет во время выполнения команды *OCW*2.

Режим спецмаскирования.

Применяется для того, чтобы программы обслуживания прерываний могли динамически изменять структуру системных приоритетов в процессе работы. Например, при выполнении какой-либо части подпрограммы обслуживания необходимо запретить запросы более низких уровней, а при выполнении другой части – разрешить их. Трудность реализации таких действий состоит в том, что пока выполняется подпрограмма и соответствующий разряд в регистре РОП не сброшен, контроллер не реагирует на запросы с более низким приоритетом. Для разрешения прерываний со всех уровней (в том числе и с более низких) устанавливается режим спецмаскирования (или затенения), запрещаются только прерывания на данном уровне. Этот режим задается и отменяется командой *OCW*3 при *SSM*=1, *SM*=1 и *SSM*=1, *SM*=0 соответственно.

Режим опроса (поллинга).

Применяется для организации обслуживания запросов прерываний по инициативе программы, выполняемой МП. При этом прерывания МП запрещаются путем сброса флага *IF* и используется команда *POLL*, которая задается установкой *P*=1 в команде *OCW*3. Тогда контроллер воспринимает следующий импульс по входу *RD* как подтверждение прерывания и выдает на ШД следующий байт: *D*7=1 – устанавливается в «1», если на вход данного контроллера поступил запрос; *D*6-*D*3 – безразличны; *D*2-*D*0=*W*2-*W*0 – задают двоичный код входа высшего уровня среди тех, по которым поступили запросы. С помощью команды *POLL* последовательно опрашиваются все контроллеры прерываний в системе, что позволяет легко наращивать число уровней приоритетов сверх 64. В этом режиме таблица адресов подпрограмм обслуживания не используется, и он полезен в тех случаях, когда одна подпрограмма применяется для обслуживания нескольких уровней прерываний.

Чтение статуса контроллера.

Программистом в программе может быть предусмотрено чтение текущих состояний (статуса) внутренних регистров РМП, РЗП и РОП каждого контроллера. Чтение регистра масок РМП осуществляется при каждом появлении сигналов *RD*=0, *A*0=1, *CS*=0 в результате выполнения команды ввода *IN*. Чтение остальных регистров предваряется засылкой команды *OCW*3 с разрядом *ERIS*=1: если разряд *RIS*=1, то по активному сигналу *RD* будет считан регистр РОП, если *RIS*=0, то – регистр РЗП. Следует отметить, что при многократном чтении одного и того же регистра РОП или РЗП повторять команду *OCW*3 нет необходимости, поскольку контроллер запоминает последнюю команду *OCW*. Напомним, что в результате инициализации контроллер настраивается на выдачу состояния регистра РЗП.

Описанные приемы чтения статуса контроллера не используются в режиме поллинга, поскольку в этом случае по сигналу *RD*=0 контроллер выдает на ШД не состояние регистров, а информацию о поступивших запросах.

3.2.4 ПРИМЕР

Пример использования контроллера прерывания в ПК: подключение клавиатуры. Нажатия (и отпускания) клавиш на клавиатуре по отношению к процессору являются асинхронными событиями. Конечно, процессор может ожидать моменты нажатия (отпускания клавиш), но это нерационально, т.к. при этом он не может эффективно или в принципе не может решать другие задачи. Поэтому медленные устройства (генерирующие небольшой объем данных на больших временных интервалах) подключаются к микропроцессорным системам посредством механизма прерываний и с использованием буферизации. Кроме того, такие устройства, как правило, содержат в себе вспомогательный контроллер.

На рис.19б представлена условная схема включения клавиатуры в ПК. Встроенный в клавиатуру контроллер преобразует события нажатия или отпускания клавиш (и их сочетаний) в так называемые скен-коды. При этом для обеспечения немедленной реакции ПК на нажатия клавиш контроллер клавиатуры вырабатывает запрос на прерывание (поступает в МП). ОС в своем составе обязательно содержит обработчик прерывания клавиатуры, задачами которого являются чтение скен-кодов из контроллера клавиатуры и отправка подтверждения приема данных. На основе скен-кодов нажатия и отпускания производится (при необходимости) их

преобразование в *ASCII* коды (в соответствии с языковыми настройками) и их сохранение во внутренний буфер системы. Также формируются флаги клавиатуры (нажатие и удерживание клавиш типа *shift*, *ctrl*, *alt*, состояние клавиш *caps lock*, *num lock* и т.п.). В свою очередь программы нуждающиеся во вводе данных с клавиатуры опрашивают этот системный буфер и флаги (получая данные о нажатых клавишах). Альтернативой ввода данных с клавиатуры служит написание собственного обработчика прерывания клавиатуры с целью перехвата скен-кодов (при этом системный обработчик либо блокируется, либо вызывается программно перед или после работы собственного обработчика).

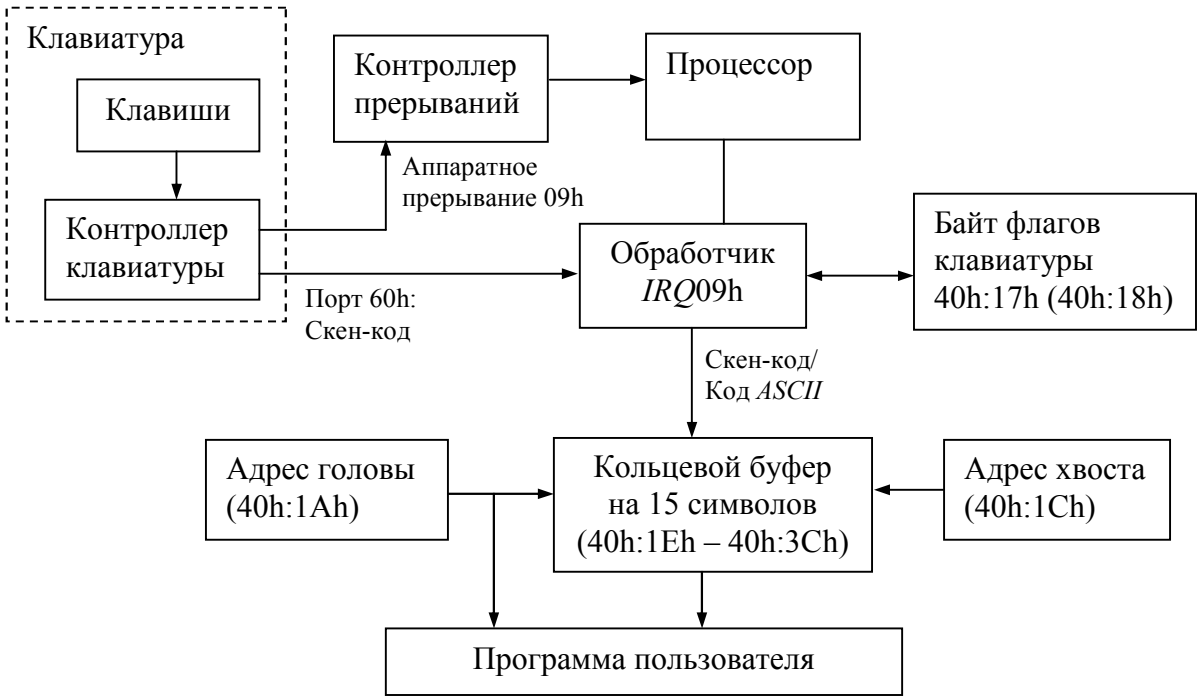


Рис. 19б – Схема включения клавиатуры в ПК на базе i8086

Назначение битов флагов клавиатуры (1 -клавиша нажата; 0 -отпущена)

40h:17h	
Бит 0	Shift правый
1	Shift левый
2	Ctrl
3	Alt
4	Scroll lock (режим)
5	Num lock (режим)
6	Caps lock (режим)
Бит 7	Insert (режим)

40h:18h	
Бит 0	Ctrl левый
1	Alt левый
2	Sys req
3	Pause (режим)
4	Scroll lock
5	Num lock
6	Caps lock
Бит 7	Insert

Программа простейшего обработчика прерывания клавиатуры:

```
Int09h proc
…
in al,60h        // чтение скен-кода

in al,61h        // генерируем строб «подтверждения» приема скен-кода
or al,80h
out 61h,al
and al,7fh
out 61h,al
mov al,20h      // неспецифическое EOI
out 20h,al   // посылка сигнала окончания прерывания в контроллер прерываний
…
Int09h endp
```

3.3 Программируемый таймер

Довольно часто требуется устройство формирования временных интервалов для процессора и внешних устройств, подсчета внешних событий и ввода показаний в процессор, а также генерирования внешней синхронизации, которую может программировать процессор. Такое устройство называется *программируемым интервальным таймером/счетчиком событий.* Некоторыми областями применения такого устройства являются:

- прерывание операционной системы с разделением времени через равномерные интервалы, чтобы она осуществляла переключение программ;
- вывод точных временных сигналов с программируемыми периодами в устройство ввода-вывода (например: в аналого-цифровой преобразователь);
- программируемая генерация скорости передачи в бодах;
- измерение временной задержки между внешними событиями;
- подсчет числа появлений событий во внешнем эксперименте и ввод показания в систему;
- прерывание процессора после появления запрограммированного числа внешних событий.

Типичная организация интервального таймера/счетчика событий показана на рис.20. Обычно доступны регистры: два верхних являются входными портами, а два нижних – выходными (относительно таймера). Сам счетчик «внешнему миру» не доступен, но может инициализироваться из регистра начального счета и считывается посредством передачи его содержимого в выходной регистр счетчика. Счетчик запускается с начального значения и отсчитывает до 0. Вход *CLK* определяет скорость счета, сигнал *GATE* разрешает и запрещает вход *CLK* и, возможно, выполняет другие функции, а выход *OUT* становится активным при достижении счетчиком 0 или, возможно, при подаче сигнала *GATE*. Выход *OUT* подключается к линии запроса прерывания в системной шине, поэтому прерывание возникает при достижении счетчиком 0; его же можно подключить к устройству ввода-вывода для инициирования необходимых действий.

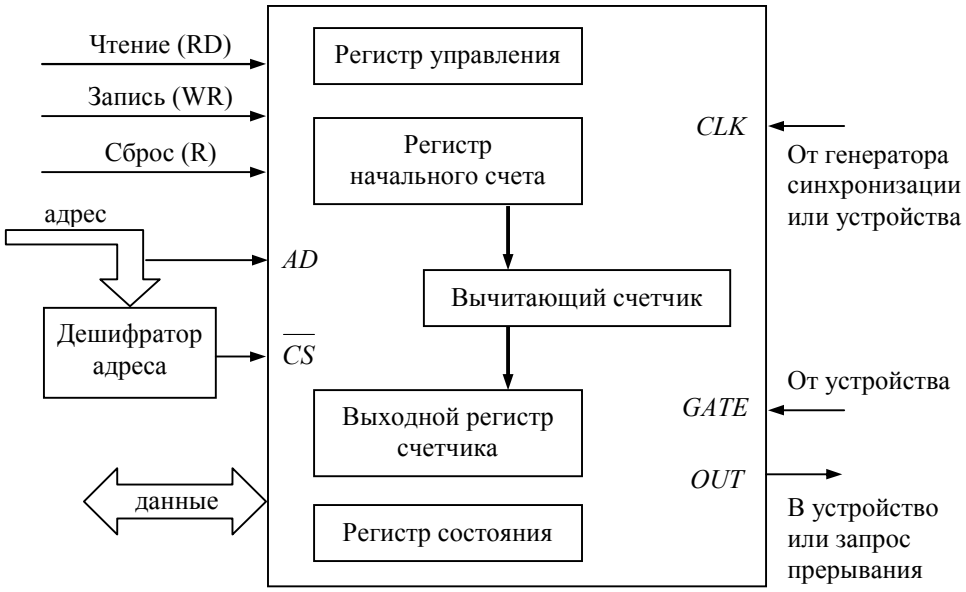


Рис. 20 – Структурная схема типичного интервального таймера/счетчика событий

Устройство вводит значение в регистр начального счета, передает его в счетчик и выполняет счет "назад" (т.е. вычитание) импульсами со входа *CLK*. Текущее содержимое счетчика в любой момент можно ввести в процессор, не нарушая работы счетчика, посредством передачи его в выходной регистр счетчика с последующим считыванием из этого регистра. При буферировании содержимого счетчика не требуется вводить его в процессор немедленно. Индикация нуля в счетчике обычно фиксируется на выходе *OUT* и в одном бите регистра состояния. Поэтому для обнаружения нуля допускается применять программный ввод-вывод и ввод-вывод по прерываниям.

Регистр управления определяет режим работы и выполняет другие функции. Режим точно определяет, что происходит при достижении счетчиком 0 и (или) при подаче сигнала на вход *GATE*. Возможными действиями являются:

- вход *GATE* применяется для разрешения и запрещения входа *CLK*;
- вход *GATE* вызывает реинициализацию счетчика;
- вход *GATE* прекращает счет и формирует высокий уровень на выходе *OUT*;
- при достижении 0 счетчик выдает сигнал *OUT* и останавливается;
- при достижении 0 счетчик выдает сигнал *OUT* и автоматически реинициализируется из регистра начального счета.

Режимы могут также определяться комбинациями перечисленных возможностей.

Рассмотрим, например, применение интервального таймера в ОС с разделением времени. В этом случае на вход *CLK* подаются сигналы синхронизации, а выход *OUT* подключается к линии запроса прерывания (возможно, немаскируемого прерывания). Вход *GATE* здесь не требуется. При включении системы в регистр начального счета необходимо загрузить значение: начальный счет =(частота синхронизации)**T*, где *T* – продолжительность каждого временного кванта в секундах. Задается такой режим, что при достижении счетчиком 0 содержимое регистра начального счета вновь загружается в счетчик, а выход *OUT* становится активным. Поскольку сигнал *OUT* используется как запрос прерывания, процедура прерывания для переключения программ будет выполняться с интервалом *T* секунд.

3.3.1 ПРОГРАММИРУЕМЫЙ ИНТЕРВАЛЬНЫЙ ТАЙМЕР 8254

Программируемый таймер (ПТ) ВИ54 предназначен для генерации времязадающих функций, программно-управляемых временных задержек с возможностью программного контроля их выполнения. Программируемые таймеры применяются в МПС, выполненных на базе БИС К580, К1810, К1821, используемых в задачах управления и измерения в реальном масштабе времени с тактовой частотой до 10 МГц. Конструктивно все эти ПТ совместимы и в частности с К580ВИ53, но отличаются от них повышенным быстродействием и расширенными функциональными возможностями.

Программируемый таймер ВИ54 включает три независимых канала, каждый из которых может быть запрограммирован на работу в одном из шести режимов для двоичного или двоично-десятичного счета. Структурная схема ПТ и его условное графическое обозначение показана на рис.21.

Назначение выводов.

CS – выборка кристалла.

Сигнал управляет входным буфером *BD*. При *CS*=0 разрешается работа буфера.

RD – чтение.

Сигнал *RD*=0 ориентирует входной буфер *BD* на вывод (ПТ выдает данные в МП).

WR – запись.

Сигнал *WR*=0 ориентирует входной буфер *BD* на ввод (ПТ принимает данные от МП).

*A*0, *A*1 – адресные входы, по которым осуществляется адресация к одному из каналов:

*A*0 *A*1= 0 0 – адрес канала 0;
*A*0 *A*1= 0 1 – адрес канала 1;
*A*0 *A*1= 1 0 – адрес канала 2;
*A*0 *A*1= 1 1 – признак загрузки управляющего слова или команд.

CLK2-CLK0 – входы тактовых сигналов для управления счетчиком/таймером. Срез сигнала на входе *CLK* приводит к уменьшению содержимого счетчика/таймера *CE* на единицу.

GATE2-GATE0 – входы разрешения счета. При *GATE*=1 разрешается выполнение функций для некоторых режимов работы разрешается поступление тактовых сигналов на вход счетчика/таймера, для других (импульсный генератор и генератор меандра) открывается выходной буфер *OUT*.

OUT2-OUT0 – выходы счетчика/таймера.

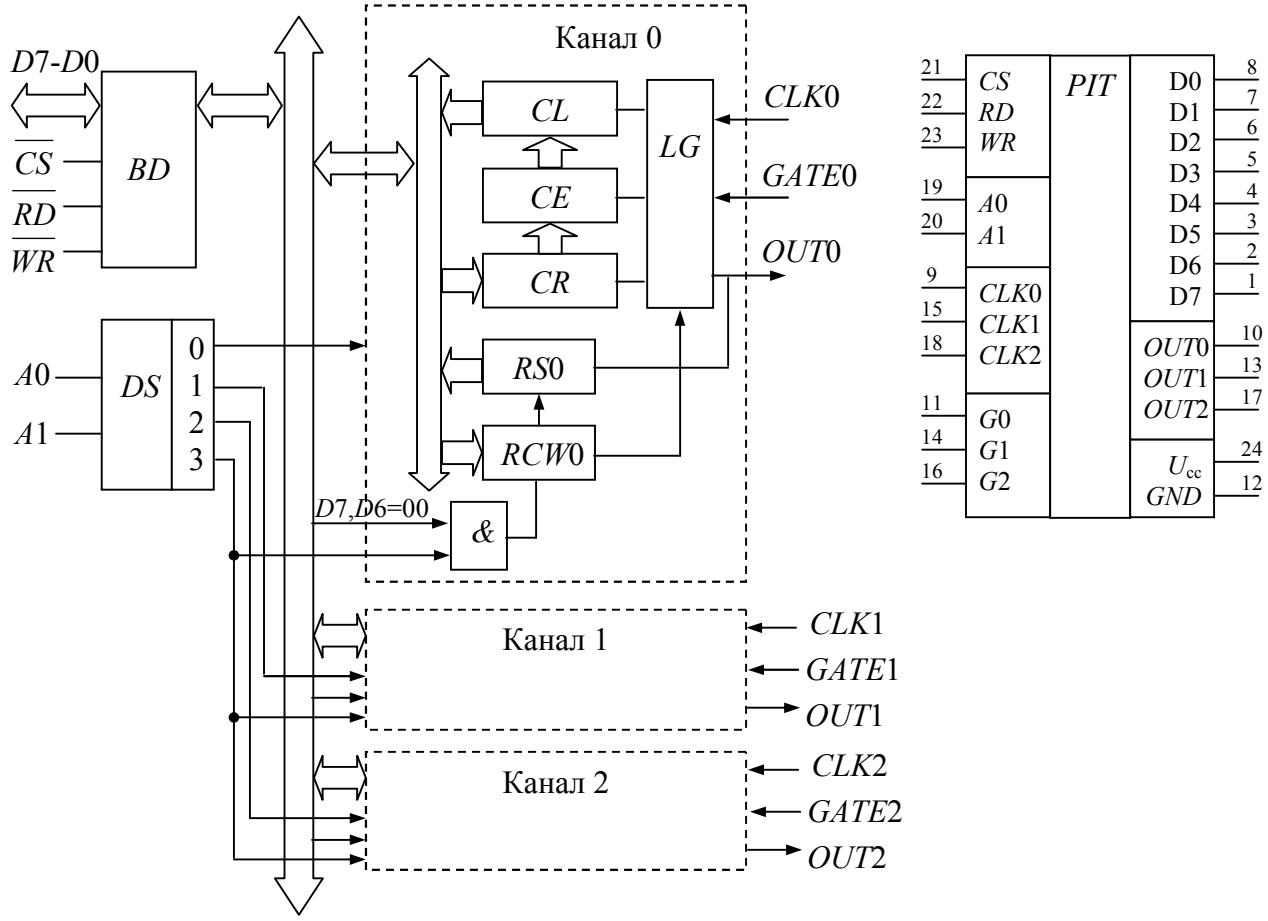


Рис. 21 – Структурная схема программируемого таймера ВИ54 и его условно-графическое обозначение

Структурная схема ПТ (рис.15) включает:

- буфер шины данных (*BD*) и логические схемы управления чтением/записью;
- дешифратор *DS*, с помощью которого выбирается один из трех каналов либо формируется признак загрузки управляющих слов или команд;
- три идентичных канала, реализующих индивидуально запрограммированную функцию.

Каждый канал включает:

1) 16-разрядный буферный регистр *CL***,** служащий для запоминания и хранения мгновенного значения счетчика *CE*, которое в любое время может быть записано командой *Защелка* или *Чтение состояния* канала. После выполнения этих команд содержимое *CL* может быть считано в ЦП без остановки дальнейшего счета в регистре *CE*;
2) 16-разрядный счетчик/таймер *CE*, работающий в режиме вычитания. Изменение содержимого CE осуществляется по срезу сигнала *CLK* при *GATE*=1;
3) 16-разрядный регистр констант пересчета *CR*, служащий для хранения констант пересчета. Содержимое *CR* загружается в *CE* для счета в зависимости от запрограммированного режима;
4) 8-разрядный регистр состояния канала *RS***,** содержимое которого можно считывать в ЦП с помощью команды *RBC – Чтение состояния канала.* Содержимое этого регистра является словом состояния канала.
5) 8-разрядный регистр управляющего слова *RCW*. Слово загружается в *RCW* командой *OUT* с адресом, формирующим на входах *A0*, *A*1 код 11. Выбор конкретного канала осуществляется с помощью двух старших разрядов самого управляющего слова.
6) Схема управляющей логики канала *LG* осуществляет управление входом/выходом счетчика/таймера в зависимости от запрограммированного режима.

По правилам загрузки счетчика/таймера *CE* содержимым регистра *CR* все шесть режимов работы ПТ можно разделить на три группы.

1. Режимы 0, 4 – режимы однократного выполнения функций. Константы из *CR* передаются в *CE* по первому тактовому сигналу *CLK* при *GATE*=1. С приходом последующих сигналов на входе *CLK* происходит уменьшение содержимого *CE*. Если во время счета на вход *GATE* подать нуль, то это приведет к останову счета. Новый положительный сигнал на *GATE* не вызывает перезагрузку счетчика/таймера, а только разрешает продолжение счета. По окончании счета выполнение действий заканчивается. При необходимости повторения функции требуется новое программирование – загрузка новой или даже той же константы.
2. Режимы 1, 5 – режимы с перезапуском. Здесь характерна возможность повторения запрограммированных функций без нового перепрограммирования. Загруженная константа сохраняется в *CR*, а ее передача в *CE* осуществляется по фронту сигнала *GATE* независимо от завершения счета.
3. Режимы 2, 3 – режимы автозагрузки. Загрузка *CE* содержимым *CR* осуществляется автоматически при выполнении условий счета (импульсный генератор и генератор меандра), поскольку это режимы с зацикливанием счета. Выход *OUT* открывается положительным сигналом на *GATE*.

3.3.2 ПРОГРАММИРОВАНИЕ ВИ54

ПТ относится к классу функционально ориентированных программно-управляемых интерфейсных БИС, поэтому перед началом работы в него необходимо загрузить управляющее слово (УС) и константу пересчета. УС задает один из шести режимов работы, тип счета (двоичный или двоично-десятичный), порядок загрузки и размерность (один или два байта) константы. Время-импульсная функция формируется на выходе *OUT* при *GATE*=1. Формирование функции осуществляется с помощью счетчика-таймера *CE*, работающего в режиме вычитающего счетчика по срезу сигнала *CLK*.

Программист может опросить состояние каналов ПТ с помощью специальных команд *Защелка (Чтение на лету)* (*CLC*) или *Чтение состояния канала* (*RBC*). Эти команды позволяют, не прерывая счета, опросить состояние счетчика/таймера *CE*. Кроме того, команда *RBC* позволяет прочитать содержимое регистра состояния канала, разряды которого несут информацию о запрограммированном режиме, состоянии выхода *OUT* и флага «обновления».

Управляющее слово *CW*.

бит 7	6	5	4	3	2	1	бит 0
*SC*1	*SC*0	*RW*1	*RW*0	*M*2	*M*1	*M*0	*BCD*

Управляющие слова загружаются в регистры *RCW* каналов ПТ по командам вывода, формирующим на входах ПТ коды *A*0*A*1=11, *CS*=0, *WR*=0, *RD*=1. Номер канала ПТ указывается в самом формате УС, и сохраняются там во все время работы или до следующего программирования. Загрузка УС должна предшествовать загрузке констант.

Поле **SC** (биты 7, 6) определяет адрес регистра *RCW* конкретного канала (*SC*1 *SC*0: 00 – канал 0, 01 – канал 1, 10 – канал 2). Если в этом поле содержится код 11, то загружаемая информация воспринимается ПТ как команда *Чтение состояния канала* (см. далее описание команды *RBC*).

Поле **RW** (биты 5, 4) определяет размерность и порядок загрузки констант. Если в поле *RW* заданы коды (*RW*1 *RW*0) 01 или 10, то размер константы определен соответственно старшим или младшим байтом. Если в поле *RW* задан код 11, то размер константы два байта; сначала загружается младший байт, затем старший. Если в поле *RW* задан код 00, то загружаемый байт воспринимается как команда *Защелка* (см. далее описание команды *CLC*).

Поле **M** (биты 3-1) задает один из шести режимов работы канала:

режим 0 (000) –прерывание от таймера;
режим 1 (001) –программируемый ждущий мультивибратор;
режим 2 (Xl0) –импульсный генератор частоты;
режим 3 (Xll) –генератор импульсов со скважностью два;
режим 4 (100) –программно-запускаемый одновибратор;
режим 5 (101) –аппаратно-запускаемый одновибратор.
В режимах 2, 3 бит 3 может принимать любое значение.

Поле **BCD** (бит 0) определяет тип счета. При *BCD*=0 константа задается в двоичном коде и может принимать значения в диапазоне 0-65535. Иначе константа задается в двоично-десятичном коде в диапазоне 0-9999.

После загрузки УС необходимо загрузить в каналы константы пересчета. Константа пересчета загружается в ПТ также по командам вывода, но с адресом, формирующим на входах *A*0, *A*1 код соответствующего канала (00, 01, 10). Константа может быть задана байтом (старшим или младшим) или 16-разрядным словом (как это определено полем *RW* управляющего слова) и представлена двоичным или двоично-десятичным кодом (как определено полем *BCD*). Порядок загрузки каналов управляющими словами и константами строго определен. Возможны два варианта. Первый предполагает загрузку в любой последовательности сначала всех УС, затем констант пересчета. Второй предполагает загрузку управляющего слова для любого канала, а затем константы пересчета для этого же канала.

Общими и обязательными требованиями для загрузки УС и констант являются следующие:

1) загрузка УС должна опережать загрузку константы;
2) загрузка констант всегда должна выполняться до конца, как определено разрядами *RW*1, *RW*0 в формате УС.

Состояние счетчика/таймера *CE* можно прочитать тремя способами.

1. Команда ввода позволяет прочесть состояние счетчика *CE* в любой момент времени (выполняется с адресом, формирующим на входах *A*0, *A*1 код соответствующего канала). Необходимым условием для выполнения этой операции является остановка счета перед выполнением команд чтения. Операцию чтения необходимо выполнять до конца, т.е. нельзя прочитать только один байт *CE*, если константа задана 16-разрядным словом. Сначала считывается младший байт, затем старший.
2. Чтение по команде *CLC* (Защелка). Команда *CLC* позволяет прочитать состояние счетчика *CE* в любой момент времени без остановки счета. Для этого необходимо загрузить эту команду в ПТ в нужный момент времени. Команда загружается в ПТ по команде вывода с адресом, формирующим на входах *A*0, *A*1 код 11. Формат команды *CLC* представлен на рис.22. Биты 7, 6 задают адрес защелкиваемого канала. Эти разряды не могут принимать значение 11, являющееся кодом команды *RBC*. Биты 5, 4 являются кодом команды *CLC* и принимают значение 00. Биты 3-0 не участвуют в операции и могут принимать любое значение.

Рис. 22 – Формат команды защелкивания *CLC*

После загрузки команды *CLC* выполняется операция чтения так же, как в первом способе, с теми же необходимыми условиями, кроме снятия сигнала *GATE*. Фактически команда *CLC* записывает состояние счетчика/таймера *CE* в буферный регистр *CL*, из которого информация считывается без нарушения продолжения счета. Информация из *CL* может быть считана в любое время. Необходимо помнить, что если операция чтения не выполнена или выполнена не до конца, то новая команда *CLC* не воспринимается.

3. Команда *RBC* (Чтение состояния канала) позволяет в любой момент времени прочитать слово состояния канала (*SW*), т.е. содержимое регистра *RS*, а также выполнить защелку одного или нескольких каналов одновременно. Формат команды *RBC* представлен на рис.23. Она загружается в ПТ так же, как УС, т.е. по команде вывода с адресом, формирующим код *A*0,*A*1=11. Старшие биты 7,6 =11 являются кодом команды *RBC*.

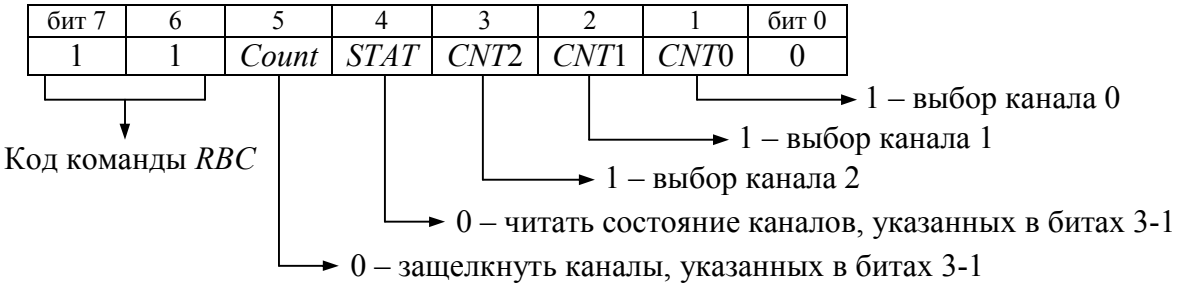


Рис. 23 – Формат команды чтения состояния канала *RBC*

Команда *RBC* может выполнять две операции:

1) *Защелку*, что аналогично команде *CLC*;
2) *Чтение регистра состояния канала.*

Эти операции задаются независимо битами 5 и 4. Возможно совмещение операций. При битах 5, 4 =11 операций нет. Особенностью этой команды является возможность выполнения операций одновременно в нескольких каналах. Установка битов 3, 2, 1 определяет операцию с соответствующими каналами.

Например, команда *RBC*=11001110 одновременно защелкивает/перезаписывает состояние регистров *CE* всех трех каналов в собственные регистры *CL*, а также состояние каналов в *RS* и делает их доступными для чтения. Эта команда эквивалентна трем командам *CLC*. Как и при команде *CLC*, новое «защелкивание» возможно только после полного считывания содержимого *CL*.

Слово-состояния канала SW. Каждый канал имеет регистр слова-состояния *RS*. Формат слова состояния показан на рис.24, из которого видно, что в процессе работы канала изменяются только два старших разряда слова состояния: биты 7 и 6. Остальные разряды соответствуют разрядам управляющего слова, что позволяет контролировать правильность его загрузки.

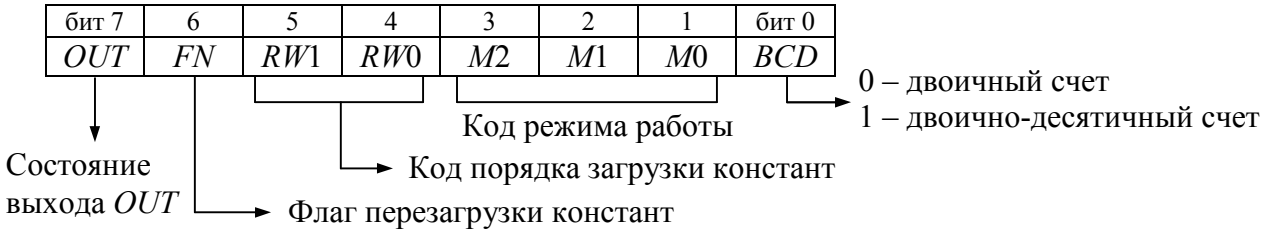


Рис. 24 – Формат слова состояния канала ПТ

OUT (бит7) слова состояния канала несет информацию о состоянии выхода канала *OUT* в момент выполнения команды *RBC*.

FN (бит 6) является флагом обновления регистра констант *CR*. Этот флаг особенно необходим для режимов 1 и 5, он позволяет определить, произошла ли загрузка константы из регистра *CR* в *CE* или нет. Напомним, что для этих режимов загрузка осуществляется аппаратно – по фронту сигнала *GATE*. Поэтому если флаг обновления установлен в нуль, то это означает, что ранее загруженная константа перезаписана из *CR* в *CE* и по ней осуществляется операция счета. В этом случае можно произвести загрузку новой константы, не нарушая предыдущего счета. По фронту следующего сигнала *GATE* начинается счет с новой константой.

Подключение ПТ к МПС осуществляется с помощью линий *CS*, *RD*, *WR*, *A*0, *A*1 по общим правилам подключения интерфейсных БИС к системной шине, в МПС на базе МП x8086 – рис.25. Входы *RD*, *WR* ориентируют входной буфер ПТ на прием или выдачу, поэтому подключаются к соответствующим выходам ЦП (*IOR*, *IOW*), на которых формируется сигнал низкого уровня при выполнении команд ввода-вывода. Вывод *CS* является входом разрешения

выбора ПТ. Нулевой потенциал на нем разрешает работу входного буфера ПТ. Этот вход используется для адресации ПТ, он подключается либо к одной из свободных шин адреса (принцип раздельной дешифрации), либо к выходу дешифратора ВУ. Выводы *A*0, *A*1 являются входами адресов внутренних регистров ПТ, они подключаются к шинам адреса, не занятым в адресации ПТ, обычно к линиям *A*0, *A*1 шины адреса МП. Временная диаграмма сигналов *CLK*, *WR*, *GATE*, *OUT* показана на рис.26, а зависимость состояний сигналов на входах ПТ и выполняемых операций ПТ с ЦП – в табл.10.

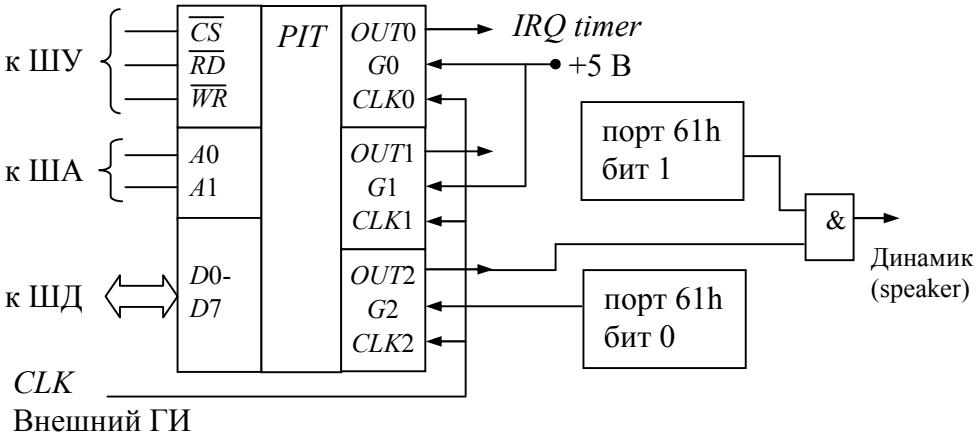


Рис. 25 – Схема включения ПТ в ПК

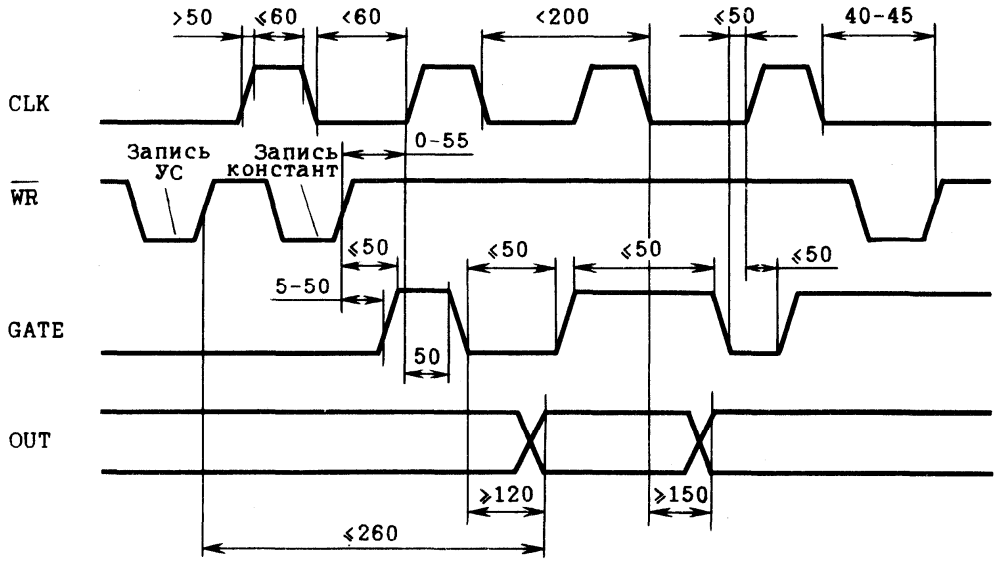


Рис. 26 – Временные диаграммы запуска канала ПТ

Таблица 10 – Управляющие сигналы ПТ ($\overline{CS} = 0$)

$\overline{RD}$	$\overline{WR}$	A_1	A_0	Операция
1	0	0	0	Запись константы пересчета в канал 0
1	0	0	1	Запись константы пересчета в канал 1
1	0	1	0	Запись константы пересчета в канал 2
1	0	1	1	Запись УС (команды)
0	1	0	0	Чтение состояния канала 0
0	1	0	1	Чтение состояния канала 1
0	1	1	0	Чтение состояния канала 2
0	1	1	1	Нет операции
1	1	x	x	Нет операции

Функционирование. Как уже отмечалось, каналы ПТ независимо могут быть запрограммированы на работу в одном из шести режимов.

В режиме 0 – *прерывание от таймера* – низкий уровень сигнала на выводе *OUT* устанавливается сразу же после загрузки УС. Загрузка константы не оказывает влияния на этот выход. Счет разрешается положительным сигналом на входе *GATE*. Изменение состояния счетчика/таймера *CE* осуществляется по срезу сигнала *CLK*, причем по первому тактовому сигналу происходит загрузка *CE* константой из *CR*, и только второй тактовый сигнал принимает участие в счете. После отсчета загруженного числа устанавливается сигнал *OUT*=1. Таким образом, сигнал *OUT*=0 удерживается на время *N*+1 тактовых периодов, где *N* – загруженная константа. Если во время счета снять сигнал *GATE*, то счет приостанавливается, содержимое счетчика/таймера сохраняется. Новый активный уровень на входе *GATE* вызывает продолжение счета без перезагрузки *CE* содержимым *CR*. Загрузка новой константы во время счета приводит: при записи младшего байта – к остановке текущего счета, а при записи старшего – к запуску нового цикла счета. Контроль счетчика (выполнение команд *CLC*, *RBC*) в этом режиме возможен только после хотя бы одного такта счета. На рис.27 показана временная диаграмма работы ПТ в режиме 0 (8 битный режим).

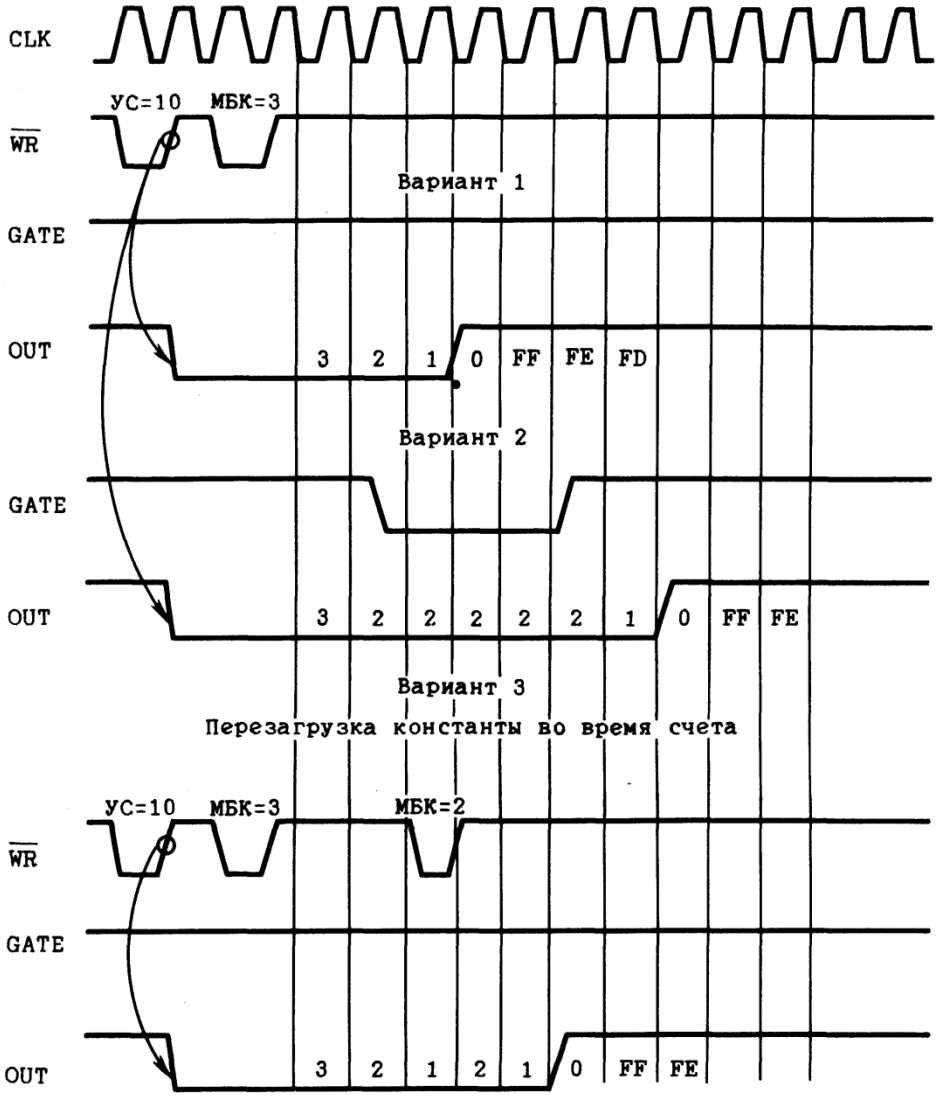


Рис. 27 – Временная диаграмма работы ПТ в режиме 0
(УС – управляющее слово, МБК – младший байт константы)

В режиме 1 – *программируемого ждущего мультивибратора* – на выходе *OUT* формируется сигнал низкого уровня длительностью $T=T_{clk}N$, где T_{clk} – период тактовых импульсов; N – константа. На выходе *OUT* по положительному фронту сигнала *GATE* устанавливается нулевой сигнал, который изменяется после окончания счета. Режим 1 является режимом с перезапуском. По каждому фронту сигнала на входе *GATE* регистр *CE* перезагружается содержимым *CR*. Это означает, что однажды загруженная константа участвует в счете всякий раз по фронту сигнала *GATE*, причем по фронту первого сигнала *GATE* флаг обновления устанавливается в нуль. Если во время счета в ПТ загружается новая константа, то она устанавливает флаг обновления в единицу, но не влияет на текущий счет. Новый счет начинается только по фронту следующего сигнала *GATE*. Выполнение команд *CLC* и *RBC* возможно только после выполнения хотя бы одного такта счета. Временная диаграмма работы ПТ в режиме 1 показана на рис.28 (8 битный режим).

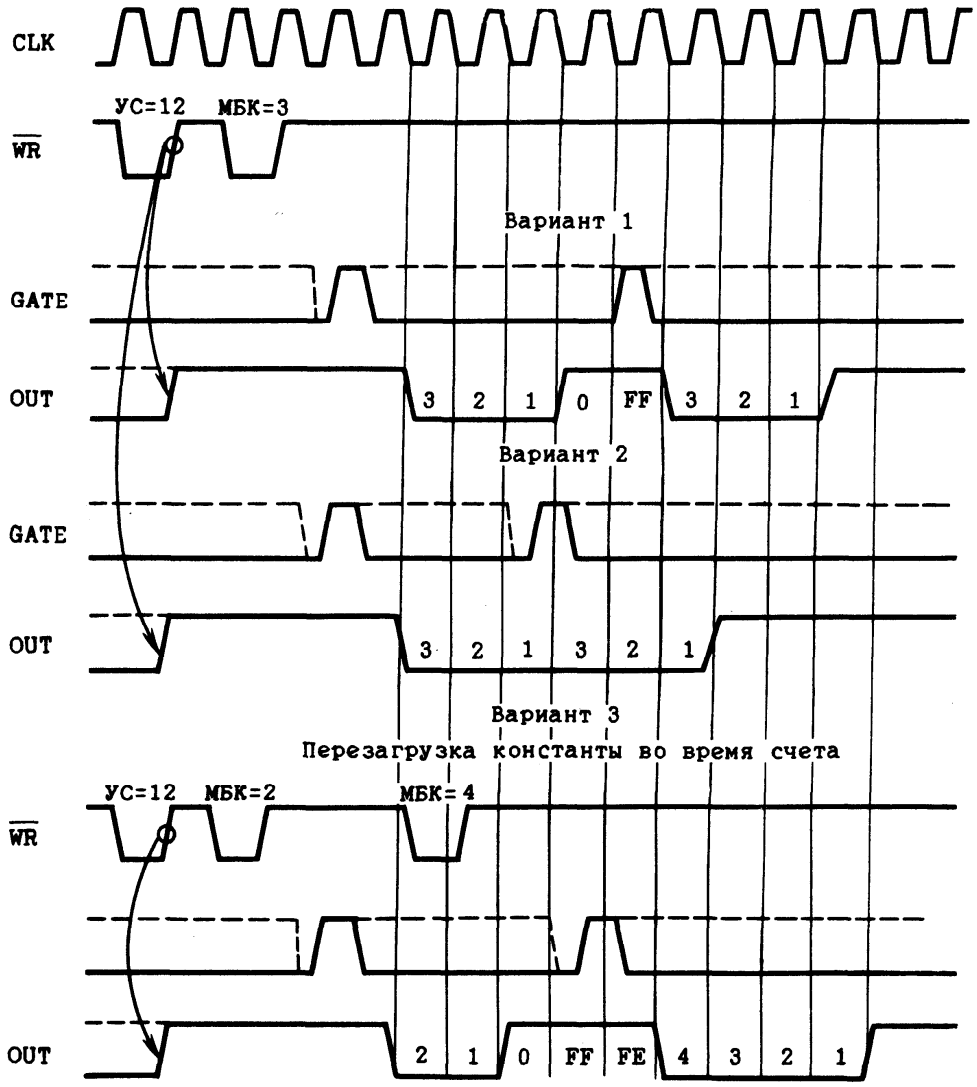


Рис. 28 – Временная диаграмма работы ПТ в режиме 1
(УС – управляющее слово, МБК – младший байт константы)

В режиме 2 – *импульсного генератора частоты* – канал работает как делитель входной частоты F_{clk} на N. Сразу же после загрузки УС на выходе *OUT* устанавливается единичный сигнал. При *GATE*=1 на выходе *OUT* с частотой F_{clk}/N устанавливается нулевой сигнал на время одного периода *CLK*. Режим 2 является режимом с автозагрузкой, т.е. после окончания цикла счета *CE* автоматически перезагружается и счет повторяется. Перезагрузка канала новой константой не влияет на текущий счет, новый счет начинается по окончании предыдущего. При *GATE*=0 на выходе *OUT* устанавливается напряжение высокого уровня и счет останавливается. При сигнале *GATE*=1 счет продолжается, что позволяет синхронизировать работу канала с внешними событиями. Выполнение команд *CLC* и *RBC* возможно для этого режима после окончания двух тактов счета. Временная диаграмма работы ПТ в режиме 2 показана на рис.29 для трех различных вариантов (8 битный режим).

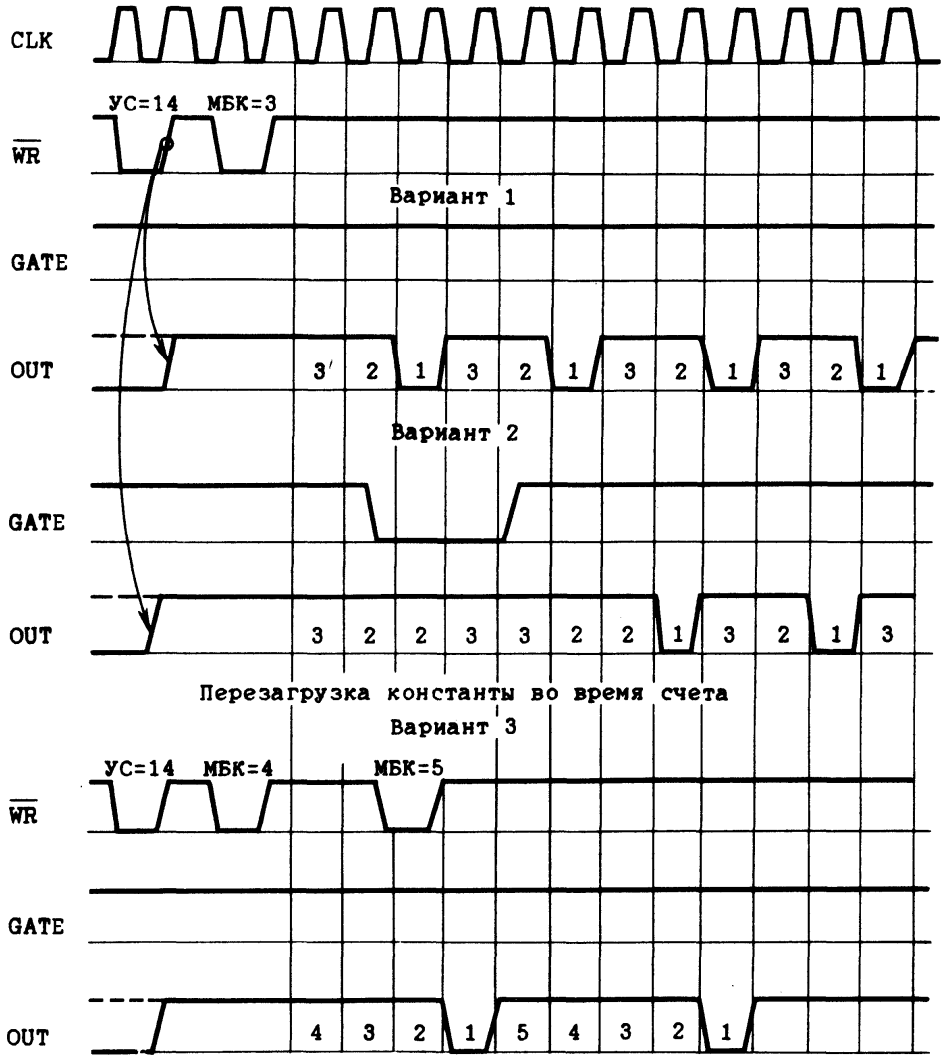


Рис. 29 – Временная диаграмма работы ПТ в режиме 2
(УС – управляющее слово, МБК – младший байт константы)

Режим 3 – *генератор импульсов со скважностью два* – аналогичен режиму 2, за тем исключением, что на выходе *OUT* формируются импульсы с длительностью полупериодов равной $(N/2)*T_{clk}$ при четных N и $((N+1)/2)*T_{clk}$ положительных и $((N–1)/2)*T_{clk}$ отрицательных полупериодов при нечетных. Этот режим является режимом с автозагрузкой, т.е. перезагрузка *CE* константой из *CR* выполняется автоматически после окончания цикла счета. Перезагрузка константы во время счета не влияет на текущий счет, новый счет начинается по окончании предыдущего. Снятие сигнала *GATE* приостанавливает счет, установка его продолжает цикл счета. В этом режиме канал может работать только с константой больше трех. Выполнение

команд *CLC* и *RBC* возможно только после двух тактов счета. Временная диаграмма работы ПТ в режиме 3 показана на рис.30 (8 битный режим).

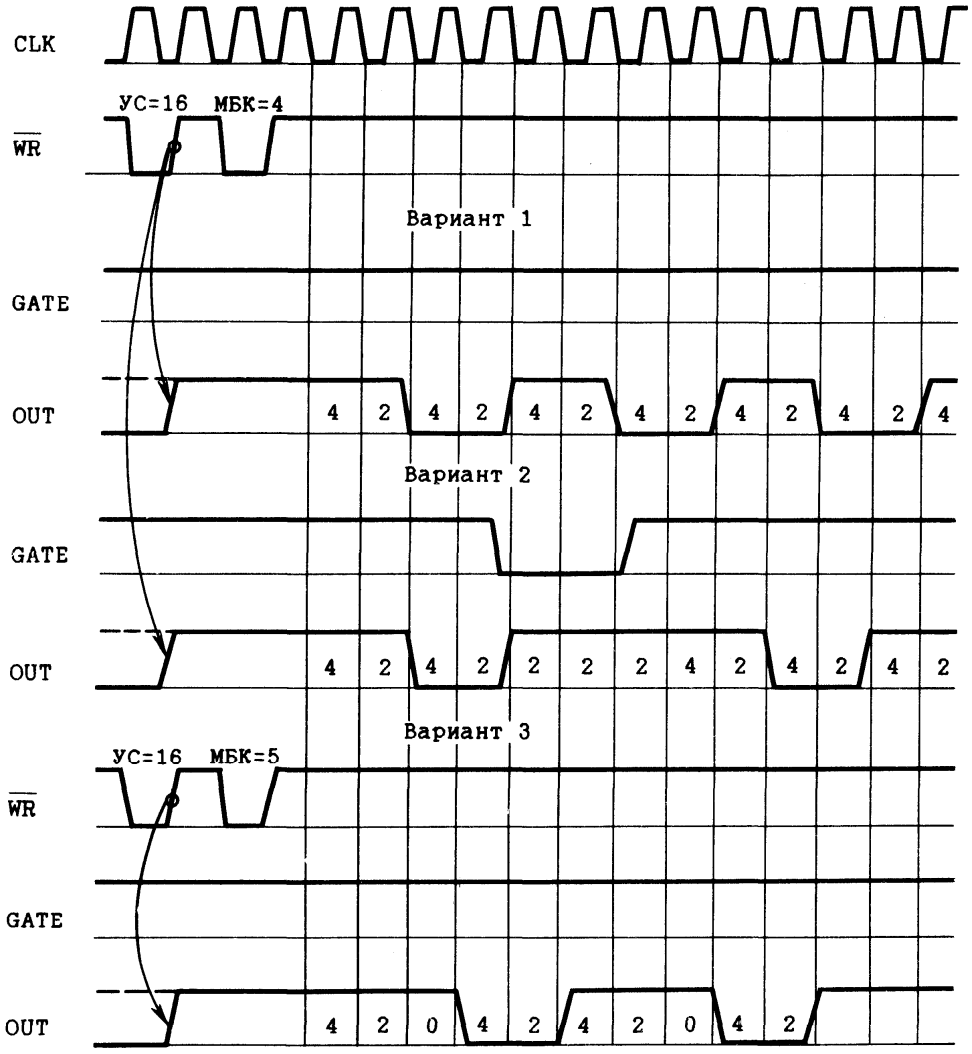


Рис. 30 – Временная диаграмма работы ПТ в режиме 3
(УС – управляющее слово, МБК – младший байт константы)

В режиме 4 – *программно-запускаемого одновибратора* – по окончании отсчета числа, загруженного в счетчик/таймер, на выходе *OUT* устанавливается нулевой сигнал на время одного периода сигнала *CLK*. Высокий уровень сигнала на выходе *OUT* устанавливается сразу же после загрузки УС. Сигнал высокого уровня на входе *GATE* разрешает счет, причем первым тактовым сигналом происходит загрузка счетчика/таймера *CE* константой из *CR*, а второй тактовый сигнал начинает счет. Таким образом, сигнал длительностью, равной периоду тактовой частоты, устанавливается на выходе *OUT* через *N*+1 тактовых периодов. Если во время счета снимается сигнал *GATE*, то счет приостанавливается, текущее значение *CE* счетчика/таймера сохраняется. Новый положительный сигнал на *GATE* вызывает продолжение счета. Это режим одноразового выполнения функции. Загрузка новой константы во время счета приводит: при записи младшего байта к остановке текущего счета, а при записи старшего — к запуску нового цикла счета. Выполнение команд *CLC* и *RBC* возможно только после окончания одного цикла счета. Временная диаграмма работы ПТ в режиме 4 показана на рис.31 (8 битный режим).

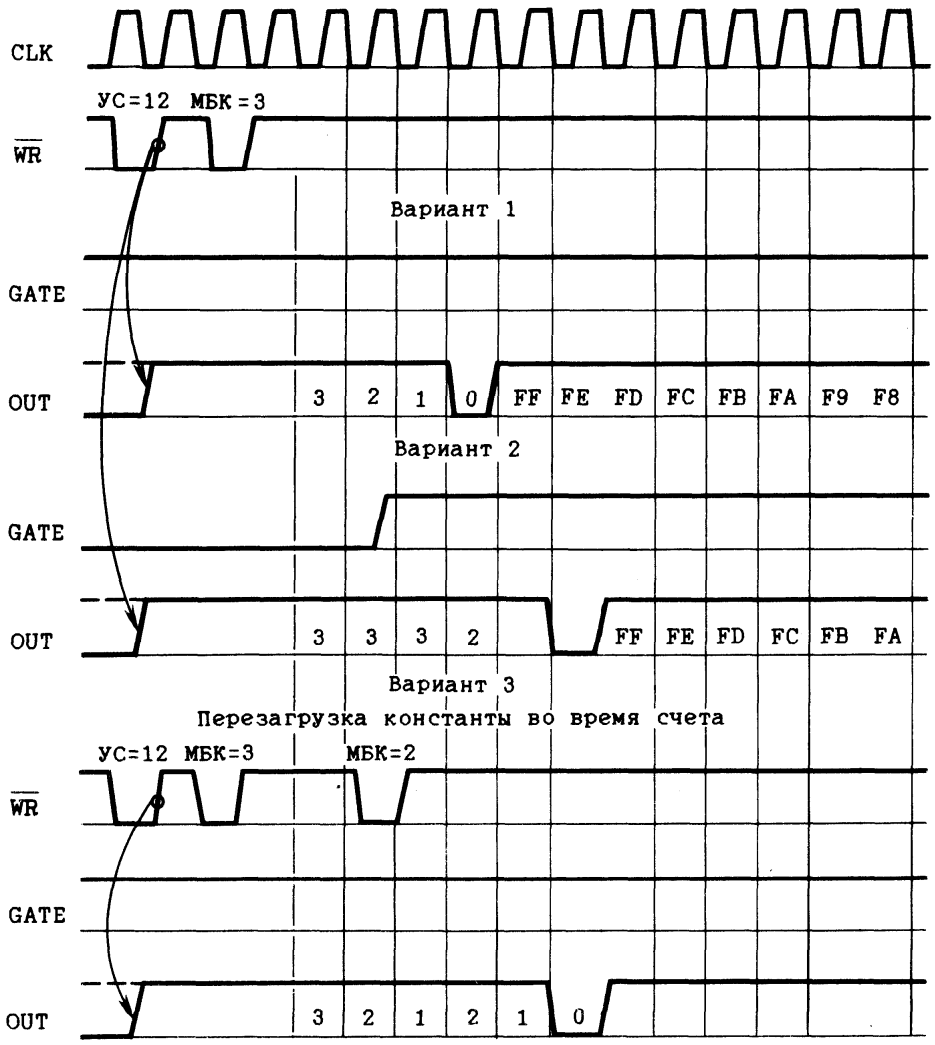


Рис. 31 – Временная диаграмма работы ПТ в режиме 4
(УС – управляющее слово, МБК – младший байт константы)

Режим 5 – *аппаратно-запускаемый одновибратор* – аналогичен режиму 4 по способу формирования сигнала на выходе *OUT* и режиму 1 по действию сигнала *GATE*. На выходе *OUT* устанавливается сигнал нулевого уровня на время одного периода *CLK* после отсчета загруженной в *CE* константы. Загрузка в *CE* константы из *CR* осуществляется по фронту сигнала *GATE*. Причем первый фронт *GATE* устанавливает флаг обновления в нуль. Если во время счета в канал загружается новая константа, то эта операция устанавливает флаг обновления в единицу, но не влияет на текущий счет. Новый счет начинается только по фронту следующего сигнала *GATE*. Выполнение команд *CLC* и *RBC* возможно только после выполнения хотя бы одного такта счета. Временная диаграмма работы ПТ в режиме 5 показана на рис.32 (8 битный режим).

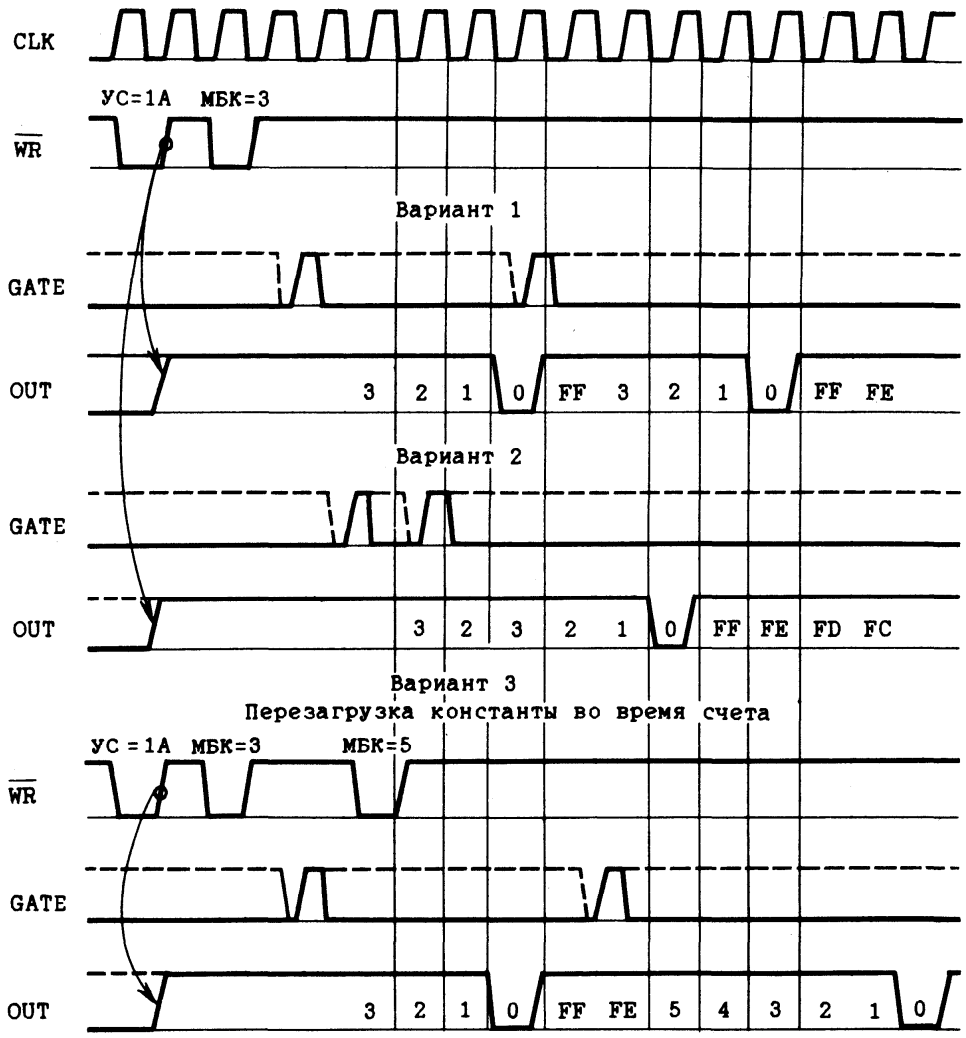


Рис. 32 – Временная диаграмма работы ПТ в режиме 5
(УС – управляющее слово, МБК – младший байт константы)

Из описания режимов ПТ видно, что таймер может реализовать все основные времязадающие функции, широко используемые в цифровой технике, с помощью которых выполняются как традиционные функции, связанные с управлением МПС, так и специфические, измерительные функции, например для измерения частоты. В заключение следует отметить, что при конструктивной совместимости таймеров К580ВИ53 и К1810ВИ54 последний кроме улучшенной частотной характеристики обладает более широкими функциональными возможностями, в особенности при реализации параллельных процессов.

3.4 DMA контроллер

Прямой доступ к памяти (ПДП) – *Direct Memory Access* – создание прямого канала обмена между внешним устройством и памятью либо обмена типа память-память.

В общем случае каждый элементарный обмен данными в один байт (слов) требует двух машинных циклов процессора. Первый – загрузка данных из источника и второй – запись этих данных в приемник.

Идея ПДП состоит в разгрузке процессора и передаче байта (слова) за один машинный цикл. ПДП особенно удобен при передаче больших блоков данных в высоком темпе. Данная задача обмена реализуется с помощью *DMA* контроллера. Взаимодействие процессора и контроллера можно проиллюстрировать рис.33.

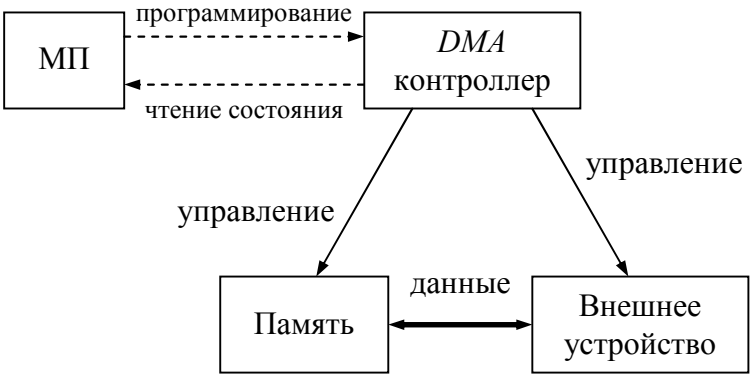


Рис. 33 – Схема взаимодействия *DMA* контроллера и процессора

МП программирует контроллер, задавая адреса приемника и передатчика, длину и кратность обмена. При начале обмена процессор отключается от шины адреса/данных, передавая управление ими контроллеру. После передачи заданного блока данных контроллер возвращает управление шиной процессору.

Возможно два подхода к передаче данных. Первый – блочный – состоит в передаче блока данных с максимальной скоростью. При этом работа процессора приостанавливается до завершения передачи. Второй – одиночная передача – состоит в передаче данных без остановки процессора, т.е. для передачи данных используются такты машинных циклов в которых процессор не использует шину (например: такт декодирования команды или такты команд не требующих обращения к памяти или устройствам ввода-вывода). В итоге производительность системы возрастает, но сам обмен с ПДП будет не быстрым, а темп нерегулярным, т.к. длительности циклов выполняемых процессором команд различны. Кроме того, ПДП в этом случае может и замедлить выполнение программы, т.к. передача слова может не уложиться во временной интервал, соответствующий «свободе» шины от запросов процессора.

На практике оказывается, что *DMA* обмен проигрывает программному обмену по множеству показателей (на *ISA* шине, а тем более *PCI*):

- по скорости передачи примерно в 2 раза (по сравнению с передачей с применением строковых команд *movs*, *ins*, *outs*);
- по простоте использования, т.к. для настройки и инициализации *DMA* контроллера необходимо выполнить серию портовых операций, в то время как при строковых командах необходимо загрузить регистры процессора;
- «свободу» процессора при *DMA* обмене трудно использовать для параллельных с обменом вычислений;
- адреса памяти при *DMA* обмене должны быть выровнены на границу 64*Kb* и 128*Kb* при 8-разрядном и 16-разрядном обмене соответственно, что приводит к неудобствам при программировании, а часто просто невозможно выделить достаточно памяти на соответствующих границах адреса.

С другой стороны, учитывая преобладание *PCI* шины (и ее клонов) с режимом *bus-mastering* необходимость в *DMA* контроллере отпадает (для совместимости современные

чипсеты эмулируют функции *DMA* контроллера). На рис.34 представлена схема включения контроллеров *DMA* (i8237) в типовой микропроцессорной системе на основе i8086.

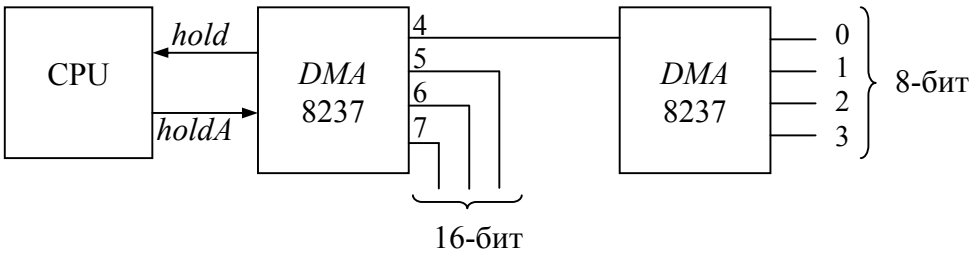


Рис. 34 – Схема включения *DMA* контроллеров в ПК

DMA контроллер обеспечивает каждый канал обмена (из 7 доступных) двумя интерфейсными сигналами: *DREQ* (запрос – выставляется устройством, желающим получить доступ к памяти), *DACK* (подтверждение – удовлетворение запроса, вырабатывается контроллером).

С программной точки зрения каждый контроллер *DMA* адресуется через порты ввода-вывода и представляется 5 регистрами: регистр маски (разрешить/запретить выбранные каналы), регистр режима/статуса (задает режим обмена), регистры страницы и указателя определяют адрес памяти, регистр длины (задает количество байт или слов для передачи).

Режимы работы контроллера

1) *Demand mode* – передача данных пока *DRQ* активно (устройство не снимает сигнал пока есть потребность в передаче) или пока не будет передано заданное количество данных (байт, слов). Если сигнал запроса *DRQ* снят до обнуления счетчика, контроллер отдает управление шине, а при последующем появлении этого запроса продолжит обмен с того места, на котором остановился.
2) *Single mode* – передача данных начинается после активации *DRQ* (устройство, получив подтверждение обмена, снимает запрос), обмен происходит до обнуления счетчика переданных данных.
3) *Block mode* – см. *Single mode*, но по окончанию обмена происходит автоматическая перезагрузка контроллера исходными (до обмена) данными. Для повторного обмена также требуется активация *DRQ*.
4) *Cascade* – режим для каскадирования контроллеров.

Типовое распределение каналов *DMA* в системе *i*8086 имеет вид:

Каналы 0, 1, 3 – доступны программисту (8 битный обмен);
Канал 2 – резервируется для *Floppy* дисков;
Канал 4 – каскадирование контроллеров;
Каналы 5, 6, 7 – доступны программисту (16 битный обмен).

3.5 Микросхема часов реального времени

В любой процессорной системе всегда существует необходимость в отсчете реального времени и ведении календаря. Данную возможность представляют микросхемы реального времени, например *MC*146818 (*Motorola*).

Микросхема *MC*146818 содержит часы реального времени и 50 байт статического ОЗУ. Микросхема выполнена по КМОП технологии и поддерживает питание от батареи для поддержания данных о времени и сохранения информации в ОЗУ при пропадании внешнего питания.

Внешний интерфейс микросхемы построен так, что при подключении к процессору не требуется ни какой внешней логики или адаптеров. Микросхема имеет следующие выводы:

Vdd – напряжение питания;

*OSC*1, *OSC*2 – входы масштаба времени, задаваемого кварцевым резонатором с частотой 4.194304 МГц или 1.048579 МГц, либо на вход *OSC*1 могут быть поданы прямоугольные импульсы с частотой 4.194304 МГц, 1.048579 МГц, 32.768 кГц;

CKOUT – выход временных импульсов;

CKFS – выбор делителя частоты, если *CKFS*=0 то частоты задаваемая на контактах *OSC* делится на 4, в противном случае – остается без изменений;

SQW – выход прямоугольных импульсов (на контакт может подаваться сигнал с одного из 15 ответвлений 22-х каскадного делителя частоты посредством программирования битов *RS*0-*RS*3 регистра *A*, включением/выключением сигнала управляет бит *SQWE* регистра *B*);

*AD*0-*AD*7 – мультиплексированная двунаправленная шина адреса и данных;

ALE – вход строба адреса *AD*0-*AD*5 (фиксирует в микросхеме адрес ячейки ОЗУ объемом 64 байта);

RD – вход строба считывания данных;

WR – вход строба записи данных;

CE – вход разрешения устройства (включение интерфейса микросхемы);

IRQ – выход запроса прерывания, сигнал активен до тех пор пока выставлен соответствующий бит состояния, вызвавший прерывание (сброс запроса осуществляется чтением регистра *C* или сигналом сброса);

PS – вход датчика наличия питания;

Reset – вход сигнала сброса, состояние которого не влияет на ход часов и содержимое ОЗУ.

При поступлении сигнала сброса происходит следующее:

a) бит периодического разрешения прерывания (*PIE*) сбрасывается на нуль
b) бит разрешения сигнализации прерывания (*AIE*) сбрасывается на нуль
c) бит разрешения прерывания по окончанию регенерации (*UIE*) сбрасывается на нуль
d) бит флага прерывания по окончанию регенерации (*UF*) сбрасывается на нуль
e) бит флага состояния запроса на прерывание (*IRQF*) сбрасывается на нуль
f) бит флага периодического прерывания (*PF*) сбрасывается на нуль
g) устройство становится недоступным
h) бит флага прерывания будильника (*AF*) сбрасывается на нуль
i) контакт *IRQ* приходит в состояние высокого импеданса
j) бит разрешения выхода прямоугольного сигнала (*SQWE*) сбрасывается на нуль

Карта памяти MC146818 показана на рис.35. Память состоит из 50 байтов ОЗУ общего назначения, 10 байтов ОЗУ, которые содержат данные времени, календаря и будильника, и четырех байтов управления и состояния. Все 64 байта могут непосредственно считываться и записываться программой процессора, за исключением:

1) регистров *C* и *D*, которые предназначены только для считывания
2) бита 7 регистра *A*, который предназначен только для считывания
3) старшего бита байта секунд, который предназначен только для считывания.

0	14	00	0	Секунды	00
13	байт	0D	1	Секунды будильника	01
14		0E	2	Минуты	02
			3	Минуты будильника	03
			4	Часы	04
	50		5	Часы будильника	05
	байт		6	День недели	06
	ОЗУ		7	День месяца	07
	пользова-		8	Месяц	08
	теля		9	Год	09
			10	Регистр *A*	0A
			11	Регистр *B*	0B
			12	Регистр *C*	0C
63		3F	13	Регистр *D*	0D

Рис. 35 – Карта адресного пространства памяти

Регистр *A*

*b*7	*b*6	*b*5	*b*4	*b*3	*b*2	*b*1	*b*0
UIP	*DV*2	*DV*1	*DV*0	*RS*3	*RS*2	*RS*1	*RS*0

UIP – бит состояния идущей регенерации (чтение). Когда *UIP*=1, цикл регенерации идет или вскоре начнется. При *UIP*=0 цикл регенерации не идет и не начнется еще по крайней мере в течение 244 мкс (для всех масштабов времени), информация о времени, календаре и будильнике в ОЗУ полностью доступна программе, т.е. нет переходного состояния. Запись "1" в бит *SET* регистра *B* запрещает любой цикл регенерации и стирает бит *UIP*.

*DV*2-*DV*0 – используются, чтобы разрешить выбор состояния 22-х каскадной цепи делителя. Биты выбора делителя определяют, какая из трех частот масштаба по оси времени используется в данный момент. Также они используются для сброса цепи делителя. Когда время и календарь инициируются впервые, программа должна запустить делитель в точное время, хранимое в ОЗУ. Когда сигнал сброса делителя снимается, первый цикл регенерации начинается на полсекунды позже. *RESET* на эти три бита не действует.

*RS*3-*RS*0 – задают частоту прямоугольного сигнала с выхода *SQW*.

Регистр *B*

*b*7	*b*6	*b*5	*b*4	*b*3	*b*2	*b*1	*b*0
SET	*PIE*	*AIE*	*UIE*	*SQWE*	*DM*	24/12	*DSE*

SET – Бит управления регенерацией (чтение/запись). Если *SET*=0, то цикл регенерации работает нормально, наращивая счет времени через каждые полсекунды. Когда бит *SET*=1, то любой идущий цикл регенерации прерывается, и программа может инициализировать байты времени и календаря без прохождения регенерации в процессе инициализации. На бит *SET* не действуют сигнал *RESET*.

PIE – Бит разрешения периодического прерывания (чтение/запись), который позволяет биту флага периодического прерывания *PF* регистра *C* вызывать переход вывода *IRQ* на низкий логический уровень. Если *PIE*=1, то вызываются периодические прерывания с частотой, указываемой битами *RS*3-*RS*0 регистра *A*. Если *PIE*=0, то *IRQ* блокируются от инициации периодическим прерыванием, но бит периодического флага (*PF*) все равно устанавливается с заданной периодической частотой. Сигнал *RESET* сбрасывает *PIE* "0".

AIE – Бит разрешения прерывания будильника (чтение/запись). Если *AIE*=1, то установка бита флага будильника *AF* в регистре *C* вызывает *IRQ*. Прерывание будильника происходит в ту секунду, в которую три байта времени равны трем байтам будильника (включая двоичный код 11xxxxxx отмены сигнала будильника). Когда *AIE*=0, бит *AF* не инициирует сигнал *IRQ*. Сигнал *RESET* сбрасывает *AIE* в "0".

UIE – Бит разрешения прерывания по окончанию регенерации (чтение/запись), который разрешает биту флага конца регенерации (*UF*) регистра *C* вызывать *IRQ*. Сигнал *RESET*=0 или установка бита *SET*=1 стирают бит *UIE*.

SQWE – Бит разрешения прямоугольного сигнала (чтение/запись). Если *SQWE*=1, то прямоугольный сигнал с частотой, указанной в битах выбора частоты (*RS*3-*RS*0) появляется на выводе *SQW*. Когда бит *SQWE*=0, контакт *SQW* удерживается на низком логическом уровне. Состояние *SQWE* стирается сигналом *RESET*.

DM – Бит режима данных (чтение/запись) показывает, в каком формате регенерируются данные времени и календаря, двоичном (=1) или двоично-десятичном (=0).

12/24 – Бит управления устанавливает формат байтов часов (чтение/запись). Если =1, то задан 24-часовое время суток или 12-часовое когда =0.

DSE – Бит разрешения перехода на летнее время (чтение/запись), позволяет совершать две специальные регенерации (*DSE*=1). В последнее воскресенье апреля время наращивается с 1:59:59AM до 3:00:00AM. В последнее воскресенье октября при достижении времени 1:59:59AM оно изменяется на 1:00:00AM. Эти регенерации не происходят, если *DSE*=0.

Регистр *C* (только чтение), все биты флагов стираются после считывания регистра *C* или когда поступает сигнал *RESET*.

*b*7	*b*6	*b*5	*b*4	*b*3	*b*2	*b*1	*b*0
IRQF	*PF*	*AF*	*UF*	0	0	0	0

IRQF – Флаг запроса прерывания устанавливается в "1", когда выполняется одно из следующих равенств: *PF*=*PIE*=1 или *AF*=*AIE*=1 или *UF*=*UIE*=1, то есть *IRQF*=*PF*&*PIE* + *AF*&*AIE* + *UF*&*UIE*. Всегда, когда *IRQF* устанавливается в "1", вывод *IRQ* переводится на низкий логический уровень.

PF – Флаг периодического прерывания. Устанавливается в "1", когда при выборе ответвления в цепи делителя достигается определенный край. Биты *SR*3-*SR*0 определяют периодическую частоту. *PF* устанавливается в "1" независимо от состояния бита *PIE*.

AF – Бит флага прерывания будильника. Если *AF*=1, то текущее время совпало с временем будильника вызывается переход *IRQ* на низкий логический уровень, но появление "1" в бите *IRQF* возможно лишь при *AIE*=1.

UF – Бит флага окончания регенерации устанавливается после каждого цикла регенерации. Когда бит *UIE*=1, то "1" в *UF* вызывает "1" в бите *IRQF*, что вызывает *IRQ*.

*b*3-*b*0 – Неиспользуемые биты регистра состояния считываются как "0".

Регистр *D*.

*b*7	*b*6	*b*5	*b*4	*b*3	*b*2	*b*1	*b*0
VRT	0	0	0	0	0	0	0

VRT – Бит достоверности данных ОЗУ и времени показывает состояние содержимого ОЗУ при условии, что контакт датчика наличия питания (*PS*) подключен. "0" в бите *VRT* появляется, если на выводах датчика наличия питания низкий логический уровень. Программа процессора может установить бит *VRT*, если время и календарь инициализированы, чтобы показать, что данные ОЗУ и времени достоверны. Бит *VRT* является битом, который не изменяется сигналом *RESET*. Бит *VRT* может быть сброшен только считыванием регистра *D*.

*b*6-*b*0 – биты не используются. Они не могут быть записаны, а считываются всегда как нули.

3.6 Последовательный интерфейс (*RS*-232)

Многие устройства ввода-вывода обмениваются информацией последовательно, т.е. по одному биту, по паре проводников, причем каждый бит занимает определенный временной интервал. Типичная конфигурация последовательного интерфейса содержит:

- регистр состояния информирует о состоянии текущей передачи (например, об ошибках)
- регистр управления задает режим работы интерфейса
- буфер входных данных подключен к регистру сдвига с последовательным входом и параллельным выходом; в операции ввода биты по одному подаются в регистр сдвига, а после приема символа информация передается в буферный регистр входных данных и ожидает передачи в ЦП
- буферный регистр выходных данных подключен к регистру сдвига с параллельным входом и последовательным выходом; вывод осуществляется выдачей данных в буфер выходных данных, передачей их в регистр сдвига и последующим сдвигом данных в последовательную выходную линию.

Если интерфейс устройства имеет отдельные линии для передачи и приема информации, т.е. для двух направлений сигналов применяются различные линии, то связь называется *дуплексной.* Такая система может передавать и принимать одновременно. В *полудуплексной* связи для ввода и вывода применяется одна и та же линия.

Имеются два основных вида последовательной связи. При *асинхронной последовательной связи* символы разделяются специальными двоичными наборами, а при *синхронной* должны быть специальные символы "синхронизации" в начале каждого сообщения и специальные "холостые" символы для "заполнения времени", когда информация не передается.

Асинхронная передача допускает любые промежутки между символами, а в синхронной передаче символы должны точно размещаться (присутствовать), даже если некоторые символы не содержат информации. Максимальная скорость передачи информации синхронной линии выше, чем асинхронной линии с той же двоичной скоростью, так как при асинхронной передаче каждый символ содержит дополнительные биты (стоповые биты, стартовые цепочки, биты четности и т.п.). С другой стороны, частоты синхронизации передатчика и приемника могут быть не точно одинаковыми (пока они находятся в допустимых пределах), так как специальные наборы допускают ресинхронизацию в начале каждого символа. При синхронной передаче действия должны быть синхронными, так как именно это определяет положение каждого бита. Следовательно, помимо данных требуется передавать сигналы синхронизации.

3.6.1 АСИНХРОННАЯ СВЯЗЬ

Формат асинхронного символа (рис.36) показывает, что символ содержит закодированную информацию и несколько дополнительных бит.

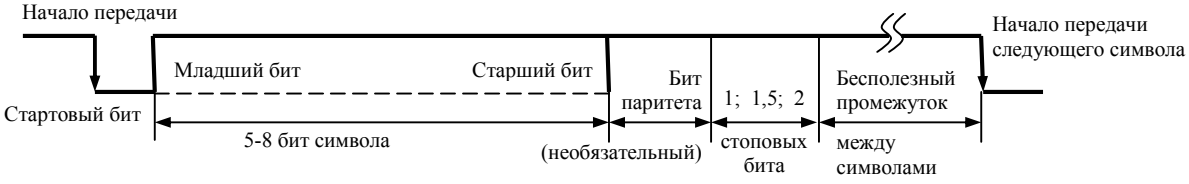


Рис. 36 – Формат стандартной асинхронной передачи

До начала передачи символа линия должна находиться в состоянии высокого уровня (лог. 1). Переход из этого состояния в состояние «0» отмечает начало символа. Первый бит всегда содержит лог. 0 и называется *стартовым битом.* Затем следуют 5-8 информационных битов, первым из которых является младший бит передаваемого символа. После информационных бит находится необязательный бит четного или нечетного паритета (по сути четность или нечетность количества единичных бит в символе). Число последних *стоповых бит* может быть 1, 1.5 или 2. Число информационных бит, тип паритета (если он есть) и число стоповых бит

могут изменяться от одной передачи (т.е. последовательности символов) к другой, но эти параметры в одной передаче являются постоянными (не изменяются).

В передатчике необходим генератор синхронизации, определяющий интервал каждого бита посредством регулирования временных отрезков между сдвигами в регистре сдвига. После выдвигания всех бит передатчик выводит лог. «1», пока не будет готов к передаче следующий символ. В приемнике также необходим генератор синхронизации для измерения интервала между сдвигами, так чтобы входной сигнал опрашивался в правильные моменты времени. Обычно частота генератора приемника в 16, 32 или 64 раза больше частоты генератора на стороне передатчика. Если множитель равен 16, после обнаружения перехода 1→0 в начале символа приемник должен отсчитать 8 импульсов синхронизации и опросить вход. При обнаружении 0 он считает, что переход вызван стартовым битом, а не помехой. Затем приемник опрашивает вход через интервалы в 16 периодов импульсов синхронизации до ввода всех бит символа, включая и стоповые биты, после чего он прекращает опрос и ожидает следующего перехода 1→0. Важно отметить, что ЦП не выдает и не принимает стартовый и стоповый биты, а также бит паритета. При выводе передатчик вводит эти биты в каждый символ самостоятельно, а при вводе приемник удаляет их из принятых данных.

Такой формат позволяет передатчику вводить между символами промежутки любой длины, а приемнику ресинхронизировать себя в начале каждого символа. Без этого механизма ресинхронизации два генератора выйдут из режима синхронизации и приемник будет опрашивать вход неправильно. При наличии ресинхронизации приемник должен учитывать скорость передачи бит только для одного символа. Неправильный опрос возникает только тогда, когда две частоты синхронизации столь различны, что сдвиг в приемнике выполняется в неверный момент времени уже через несколько бит после стартового бита. Если же это происходит, появляется большая вероятность того, что приемник обнаружит нулевой бит на месте стопового. Когда вместо стопового бита обнаруживается «0», возникает так называемая *ошибка кадра.* Таким образом, большинство асинхронных последовательных интерфейсов должны обнаруживать три вида ошибок: паритета, кадра и перегрузки (принят следующий байт когда предыдущий еще не считан).

Сигналы синхронизации передачи и приема (при дуплексной связи), которые определяют временные соотношения в интерфейсе, не обязательно должны быть одними и теми же и не обязательно должны иметь в точности одинаковую (кратную) частоту.

Устройства, которые выполняют прием и передачу данных в формате рис.36 (т.е. выполняют параллельно-последовательное и последовательно-параллельное преобразования, обнаруживают ошибки паритета, перегрузки и кадра), *называются универсальными асинхронными приемопередатчиками* (УАПП или *UART*). Микросхемы УАПП выпускаются в виде законченных и универсальных интерфейсов и их протоколы, электрические интерфейсы стандартизованы. Иногда УАПП является лишь небольшой частью интерфейса, который может реализовать большинство видов связи.

3.6.2 СИНХРОННАЯ СВЯЗЬ

Передаваемый синхронно символ также состоит из 5-8 бит с необязательным битом паритета, но не имеет стартового и стоповых бит. Все символы содержат одинаковое число бит (n) и время передачи разделяется на интервалы из n бит каждый. Опросом в приемнике управляет та же самая синхронизация, которая применяется для генерирования бит, что гарантирует синхронность двух процессов. Передатчик должен передавать символ в течение каждого n-битного интервала. Если символ к началу интервала отсутствует, возникает недогрузка и передатчик вводит холостой символ. В зависимости от системы приемник может интерпретировать холостые символы как *ошибки недогрузки.*

Проблемы запуска или коммутации в асинхронной системе не существует, так как передатчик просто выдает в линию маркер (лог. 1) до готовности начала передачи. Если помеха вызывает случайный переход 1→0, фиксируется ложный сигнал, когда первый опрос не обнаруживает стартового бита, и система ожидает следующего перехода 1→0. В синхронной передаче проблема запуска после включения или другого нарушения оказывается более сложной. Все передачи должны начинаться с серии символов синхронизации, которые нельзя

спутать с другими символами. Обычно они совпадают с холостыми символами; в коде *ASCII* символ синхронизации кодируется как 0010110.

Приемник, который должен знать код символа синхронизации, проверяет каждый бит по мере его появления и, когда последовательность бит точно соответствует битам в символе синхронизации, полагает, что началась передача. Затем он считает следующие символы передаваемой информацией или пытается соотнести один или несколько из них с символом синхронизации. Так как помеха может вызвать ложную идентификацию символа синхронизации, в большинстве систем требуется, чтобы передача начиналась серией символов синхронизации. В этом случае приемник не фиксирует начала передачи до поступления нужного числа символов синхронизации. Ненужные холостые символы и символы синхронизации удаляет приемник или программа ввода.

3.6.3 ПРОГРАММИРУЕМЫЙ СВЯЗНОЙ ИНТЕРФЕЙС

Как пример устройства последовательного интерфейса, рассмотрим микросхему 8251A программируемого связного интерфейса, структурная схема которой представлена на рис.37.

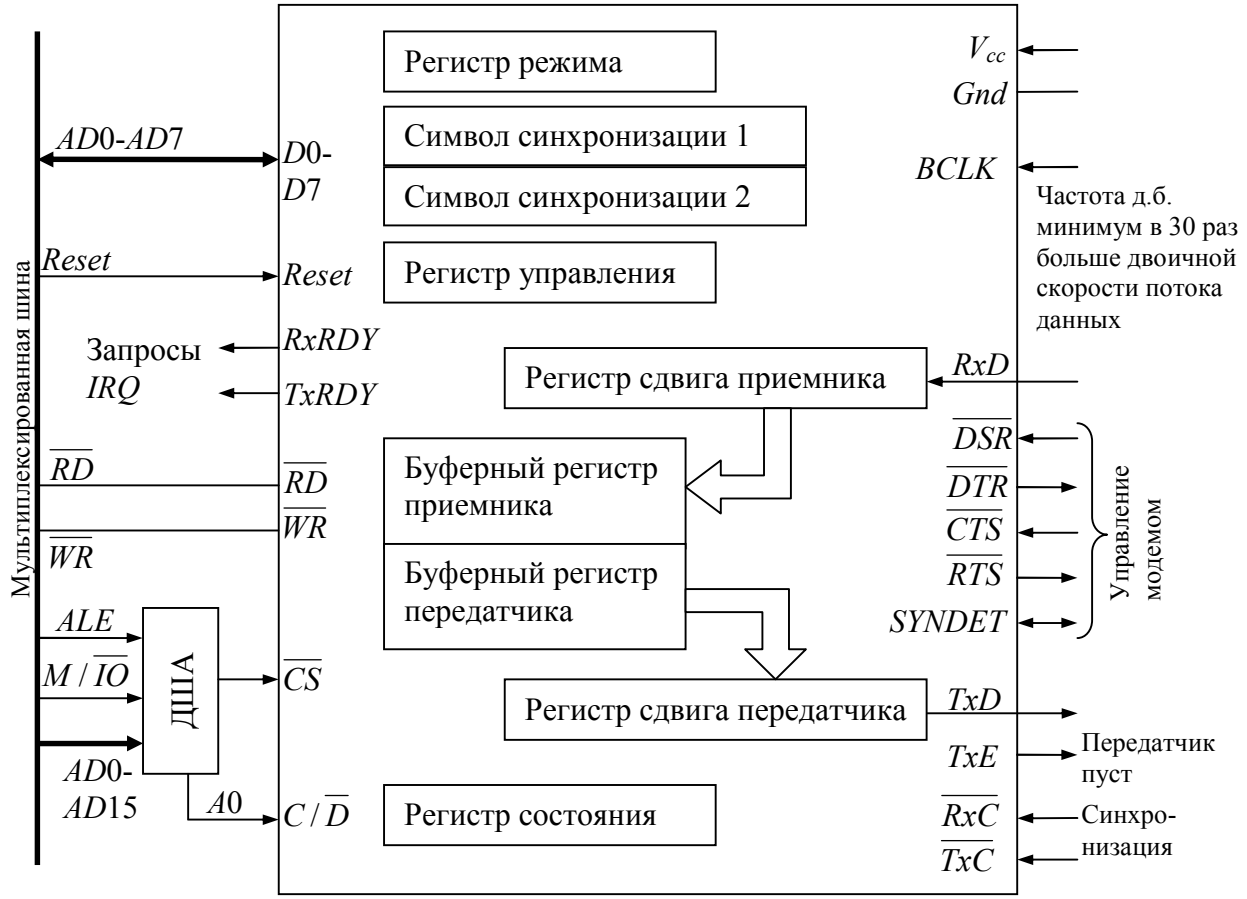


Рис. 37 – Структурная схема связного интерфейса

Микросхему можно запрограммировать для асинхронной или синхронной передачи данных. Буферные регистры приемника (входных) и передатчика (выходных) данных разделяют порт с одним и тем же адресом. Последовательный двоичный поток со входа *RxD* вводится в регистр сдвига приемника, а затем только биты данных (без служебных битов) передаются в буферный регистр входных данных, а оттуда могут считываться через шину в МП. Выводимые данные МП помещает в буферный регистр выходных данных, затем они передаются в регистр сдвига передатчика и вместе с битами синхронизации – на выход *TxD*. Содержимое регистра управления определяет синхронный или асинхронный режим работы интерфейса и формат передаваемых/принимаемых символов, а регистр состояния обеспечивает программе информацию о текущем состоянии интерфейса. Для синхронной связи необходимы

регистры для хранения символов синхронизации, в этой микросхеме их два (в асинхронном режиме они не задействованы).

Семь регистров микросхемы адресуются через два порта ввода-вывода (в адресном пространстве устройств МП резервируется два смежных адреса). Вход *C*/*D* подключается к линии адреса *A*0 и дифференцирует адреса двух портов. Микросхема интерпретирует сигналы *C*/*D, RD* и *WR* в соответствии с табл.11.

Таблица 11. Интерпретация сигналов управления

$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	Действие
0	0	1	МП читает данные из буфера входных данных
0	1	0	МП записывает данные в буфер выходных данных
1	0	1	Передача регистра состояния на шину
1	1	0	Загрузка с шины данных в регистры режима, управления или символа синхронизации

(*) Остальные комбинации сигналов переводят выходы *D*7-*D*0 в высокоимпедансное состояние.

Выбор регистров режима, управления или символа синхронизации зависит от последовательности обращения, схема которой представлена на рис.38.

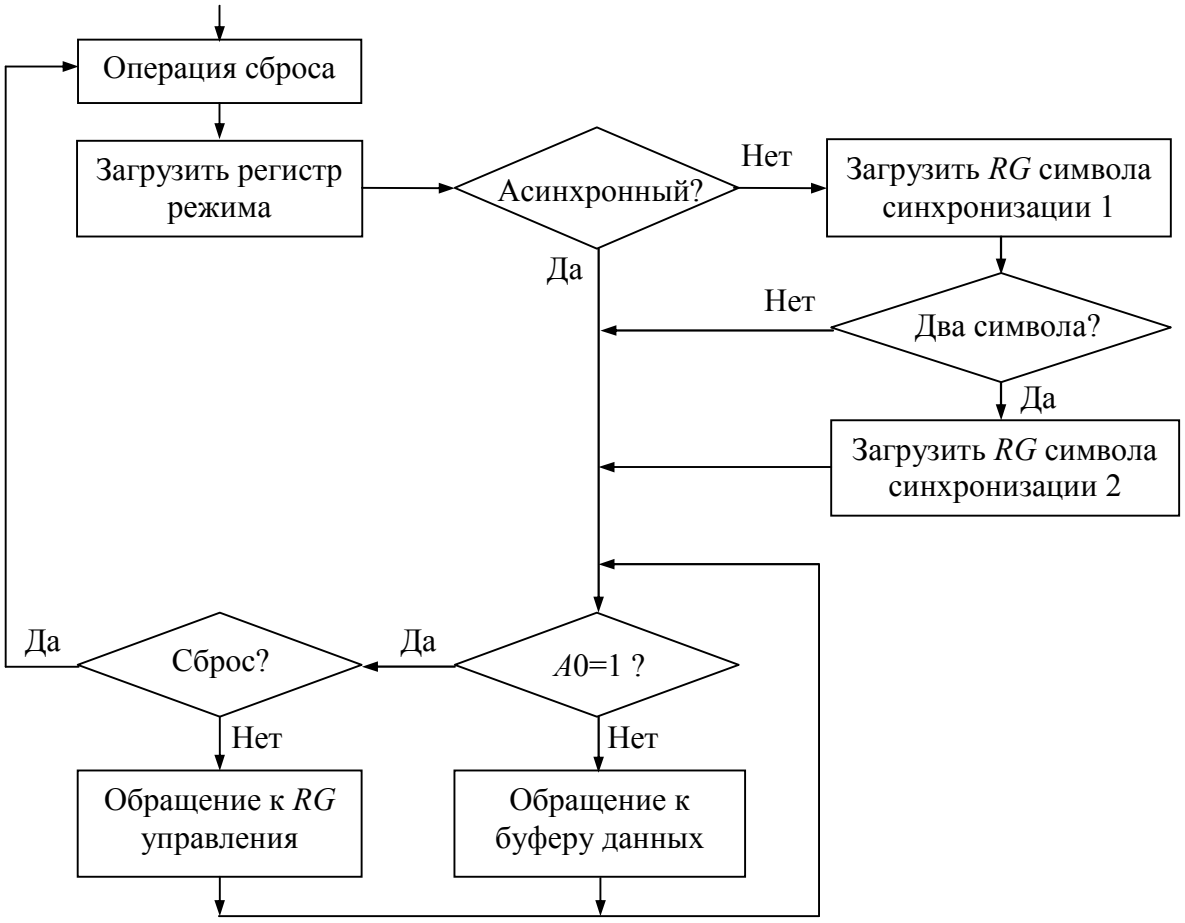


Рис. 38 – Алгоритм программирования интерфейса

После аппаратного сброса или команды с установленным в 1 битом сброса следующий вывод с *A*0=1 (т.е. *C*/*D*=1, *RD*=1 и *WR*=0) предназначен для регистра режима. Форматы регистра режима для асинхронного и синхронного случаев показаны на рис.39. Если два младших бита режима содержат нули, интерфейс переводится в синхронный режим и старший бит определяет число символов синхронизации. При этом следующие один или два байта, выводимые с *A*0=1, становятся символами синхронизации. Если два младших бита режима не равны 0, интерфейс работает в асинхронном режиме. В любом случае все последующие байты до другого сброса направляются в регистр управления (если *A*0=1) или в буферный регистр выходных данных (если *A*0=0).

Рис. 39 – Формат регистра режима: а) асинхронный; б) синхронный

В синхронном режиме скорости в бодах передатчика и приемника, равные частотам сдвига регистров сдвига, совпадают с частотами сигналов на входах *TxC* и *RxC* соответственно. При работе в асинхронном режиме три оставшиеся комбинации младших бит в регистре режимов определяют множитель скорости. Взаимосвязь между частотами *TxC* и *RxC* входов синхронизации и скоростями в бодах передатчика и приемника имеет следующий вид: частота синхронизации = множитель * скорость в бодах. Например, если младшие биты регистра режима содержат 10 и скорости в бодах передатчика и приемника равны 300 и 1200, то частота на входе *TxC* должна быть 4800 Гц, а на входе *RxC* – 19.2 кГц.

В обоих режимах биты 2 и 3 показывают число информационных бит в каждом символе, бит 4 определяет наличие или отсутствие бита паритета, а бит 5 – тип паритета (нечетный или четный). В асинхронном режиме два старших бита показывают число стоповых бит. Для синхронного режима бит 6 показывает, каким является сигнал *SYNDET*: входным или выходным, а бит 7 задает число символов синхронизации. Если контакт *SYNDET* служит выходом, его сигнал становится активным, когда обнаружено соответствие входного двоичного потока и символа синхронизации. Когда же поиск символов синхронизации осуществляется внешним устройством, на вход *SYNDET* подается сигнал о фиксации соответствия из вне. Этот же контакт, но только в качестве выхода применяется и при асинхронной работе – сигнал на этом выходе называется сигналом обнаружения разрыва и он высоким уровнем отмечает прием символа, состоящего из одних нулей.

Формат регистра управления приведен на рис.40.

Бит 0 (*TxEN*) должен содержать «1» до вывода данных, а бит 2 (*RxE*) должен быть в состоянии «1» до приема данных.

Программный ответ модему осуществляется установкой в «1» бита 1 (*DTR - data terminal ready* – готовность выходных данных), так как при этом на выходе *DTR* формируется «0», а инверсия *DTR* обычно подключается к линии *CD* модема.

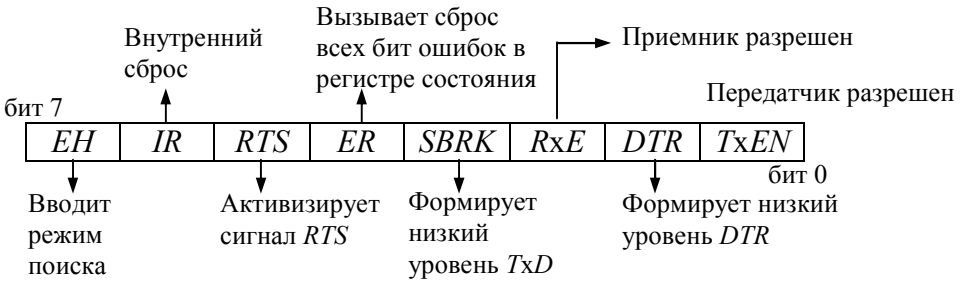
Состояние «1» бита 3 (*SBRK*) формирует *TxD*=0, вызывая передачу символов разрыва.

Установка в «1» бита 4 (*ER*) вызывает сброс всех бит ошибок в регистре состояния.

Бит 5 *RTS* (*request to send*) применяется для выдачи сигнала "запрос передачи" в модем.

Установка бита 6 (*IR*) в «1» вызывает реинициализацию 8251A и переход к последовательности сброса (т.е. осуществляется возврат в начало схемы на рис.38 и следующий вывод производится в регистр режима).

Бит 7 (*EH*) применяется только в синхронном режиме; установка его в «1» заставляет 8251A начать поиск символа (ов) синхронизации.



(*) Установка единицы в разряде включает указанную функцию

Рис. 40 – Формат регистра управления

Формат регистра состояния приведен на рис.41. Биты 1, 2 и 6 отражают состояния сигналов *Rx**RDY*, *Tx**E* и *SYNDET*. Бит *Tx**RDY* показывает, что буфер выходных данных пустой. В отличие от контакта *Tx**RDY* на этот бит не влияет входной сигнал *CTS* (*clear to send* – сброс для передачи) или бит управления *Tx**EN*. Бит *Rx**RDY* показывает, что символ принят и готов для ввода в МП. Биты *Tx**RDY* и *Rx**RDY* можно использовать для программного ввода-вывода, а сигналы с соответствующих контактов можно подключить к линиям запросов прерываний для организации ввода-вывода по прерываниям. Бит *Tx**RDY* автоматически сбрасывается, когда имеется символ для передачи, а бит *Rx**RDY* – когда установивший его символ считан процессором. Бит 2 показывает, что регистр сдвига передатчика ожидает загрузки символа из буферного регистра выходных данных. В синхронной передаче, пока этот бит установлен, передатчик будет брать данные из регистров символов синхронизации до загрузки данных в буферный регистр выходных данных. Биты 3, 4 и 5 показывают ошибки паритета, перегрузки и кадра. Когда обнаруживается ошибка, соответствующий ей бит устанавливается в «1». Если инверсию входа *DSR* (*data set ready* – готовность данных) подключить к линии *CC* модема, то бит 7 отражает его состояние – он устанавливается в «1», когда модем включен и находится в рабочем режиме.

(*) Установка единицы в разряде указывает на включение указанной функции

Рис. 41 – Формат регистра состояния

3.7 Подключение внешних устройств к шине процессора ВМ86

Для подключения любого внешнего устройства к процессору необходимо:

1) выделить этому устройству адрес (или диапазон адресов) в пространстве адресов памяти, и/или в пространстве адресов портов;
2) построить дешифратор этого адреса (либо диапазона адресов);
3) предусмотреть аппаратные средства для демультиплексирования шины;
4) предусмотреть при необходимости необходимую логику для преобразования служебных сигналов и формирования временных диаграмм элементов подключаемого устройства;
5) предусмотреть сброс устройства при появлении сигнала *Reset.*

Рассмотрим задачу подключения к шине процессора двух светодиодов (зеленого и красного) и кнопки, нажатие на которую вызывает немедленное выключение светодиодов. Для решения данной задачи выберем порт 1000h. И условимся, что младший бит (*b*0) будет управлять зеленым светодиодом, следующий бит (*b*1) – красным. При этом высокий уровень сигнала соответствует включенному светодиоду, а низкий – выключенному. Тогда принципиальная схема устройства имеет вид – рис.42.

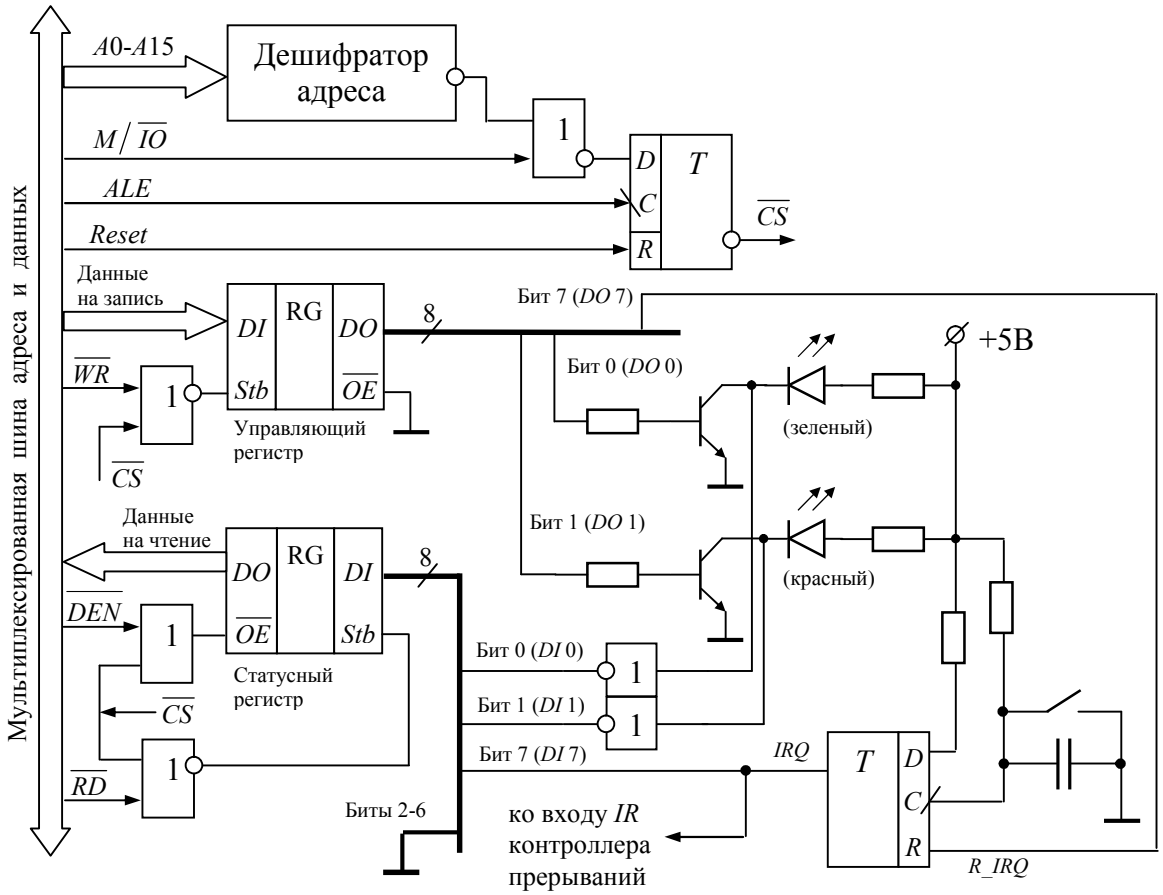


Рис. 42 – Схема устройства (пример)

Для реализации буферов данных (управляющий и статусный регистры) можно применять регистры типа К1810ИР82 или аналогичные. В совокупность логических элементов, триггеров реализуется на основе элементов стандартной серии.

Задачей дешифратора адреса является выработка строба при поступлении на его входы определенной двоичной комбинации логических нулей и единиц за фиксированный интервал времени (длительность строба *ALE*). Дешифратор может быть выполнен любым способом:

- применением ПЛИС (ПЛМ, ПМЛ и т.п.);
- на дискретных логических элементах подходящей серии;
- комбинаторных элементах таких как демультиплексор, дешифратор и т.п.;
- применением элементов с открытым коллектором, образующим монтажное «и».

Рационально применение программируемой логики для реализации всех "нужд" задачи (построение дешифратора, реализация триггеров, регистров, логических элементов).

Процедура обработки прерывания:

1) установка 7-го бита управляющего регистра устройства
2) сброс 7-го бита управляющего регистра устройства
3) посылка команды *EOI* в контроллер прерываний
4) выключение светодиодов (сброс битов 0 и 1 регистра управления)
5) возврат

Программа обработки *IRQ* (для определенности считаем, что устройство подключено к ведущему контроллеру)

```
irq_xx proc
  push AX
  push DX
  mov DX,1000h      //адрес устройства
  mov AL,128        //установка бита 7 и сброс битов 0,1
  out DX,AL         //снимаем запрос IRQ и выключаем светодиоды
  xor AX,AX         //AL=0
  out DX,AL         //сбрасываем 7 бит регистра управления, устройство готово
                    //к следующему циклу работы
  mov AL,20h        //код команды EOI - неспецифическое окончание прерывания
  out 20h,AL        //запись по адресу контроллера прерываний рабочей команды OCW2
  pop DX
  pop AX
  iret
irq_xx endp
```

Выводы

1) выделили устройству ресурсы: адрес (порт) и линию прерываний;
2) построили дешифратор адреса;
3) выполнили исполнительную часть и согласовали ее с шиной МП, т.е. буферизировали потоки данных;
4) выполнили необходимые преобразования управляющих сигналов шины для формирования требуемой временной диаграммы внешнего устройства;
5) учли нагрузочную способность шины и логических элементов, а также выполнили требования на быстродействие системы – один цикл микропроцессора, необходимо еще позаботиться о сигнале готовности (обеспечить его уровень: всегда готов с помощью элемента с тремя состояниями);
6) сделали программную часть.

4. Защищенный режим (*IA*-32)

Процессоры семейства x86, начиная с i386, могут работать в трех режимах: реальный (полностью совместимый с i8086), защищенный (позволяет реализовать максимальные возможности процессора) и виртуальный (i8086 процессор с увеличенным быстродействием и расширенными возможностями). Кроме того, имеется еще «*system management mode*», что можно перевести как режим системного управления. А позже от виртуального режима фактически попытались отказаться (выпускались процессоры без него).

На сегодняшний день архитектура под обобщающим названием *IA*-32 подразумевает:
- аппаратная адресация памяти по 32-разрядным регистрам;
- мультизадачность (аппаратная поддержка выполнения нескольких программ одновременно);
- защита программ (от воздействия друг на друга, умышленного и случайного);
- обработка прерываний (исключений);
- мультипроцессорность (аппаратные механизмы взаимодействия в системах с многими *CPU*);
- система кэширования (программная составляющая управления кэшами);
- распределение ресурсов;
- управление энергопотреблением;
- аппаратный мониторинг показателей производительности;
- аппаратная поддержка «*debugging*» (отладка программ).

Все процессоры начинают работу в реальном режиме (после включения или поступления сигнала *Reset*), а затем загрузчик ОС переводит процессор чаще всего в защищенный режим (это касается и 64-разрядных процессоров и ОС, но есть свои особенности).

Назначением защищенного режима является обеспечение независимого выполнения нескольких задач (программ), что подразумевает защиту ресурсов одной задачи от воздействия на них другой задачей. В число таких ресурсов входит память, каналы ввода-вывода и прерывания. Одним из преимуществ перед реальным режимом является механизм адресации. В защищенном режиме адресуемое пространство (при 32-х разрядном адресе) достигает 4 Гбайт физической памяти и 64 Тбайт виртуальной памяти. В этом режиме процессор кроме обычных регистров, которые являются теперь 32-разрядными (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, EIP, EFLAGS), и сегментных по-прежнему 16-разрядных (CS, DS, ES, SS) имеет два дополнительных сегментных регистра FS, GS (также 16-разрядных) и несколько дополнительных регистров служебного назначения:
- управляющие 32-х разрядные регистры *CR*0-*CR*4;
- системные адресные регистры *GDTR* (*global descriptor table RG*), *LDTR* (*local descriptor table RG*), *IDTR* (*interrupt descriptor table RG*), которые включают в себя атрибуты, 32-разрядный линейный адрес и 16-битный лимит (размер) таблицы указателей;
- регистр задачи *TR* (*task RG*), определяющий текущую задачу;
- регистры отладки (*debug RG*): *DR*0-*DR*3 – 32-х разрядные точки останова; *DR*4-*DR*5 – не используются; *DR*6-*DR*7 – регистры состояние и управления точками останова;
- регистры тестирования, количество которых зависит от типа процессора, позже эти регистры войдут в число модельно-специфических регистров *MSR* (*model-specific RG*), которые предназначены для управления отладкой, мониторингом производительности, кэшированием областей памяти и т.п.

Структурная схема процессора (системный уровень) с точки зрения архитектуры *IA*-32 представлена на рис.43. Каждая задача (*task*) содержит участки памяти с кодом, данными и стеком. Особенности есть у обработчиков прерываний и исключений, которые используют стек прерываемой задачи.

Память для 32-х разрядных процессоров 80x86 подразделяется на байты (8 бит), слова (16 бит), двойные слова (32 бита). Слова и двойные слова записываются в смежных байтах, начиная с младшего. Адресом слова или двойного слова является адрес его младшего байта.

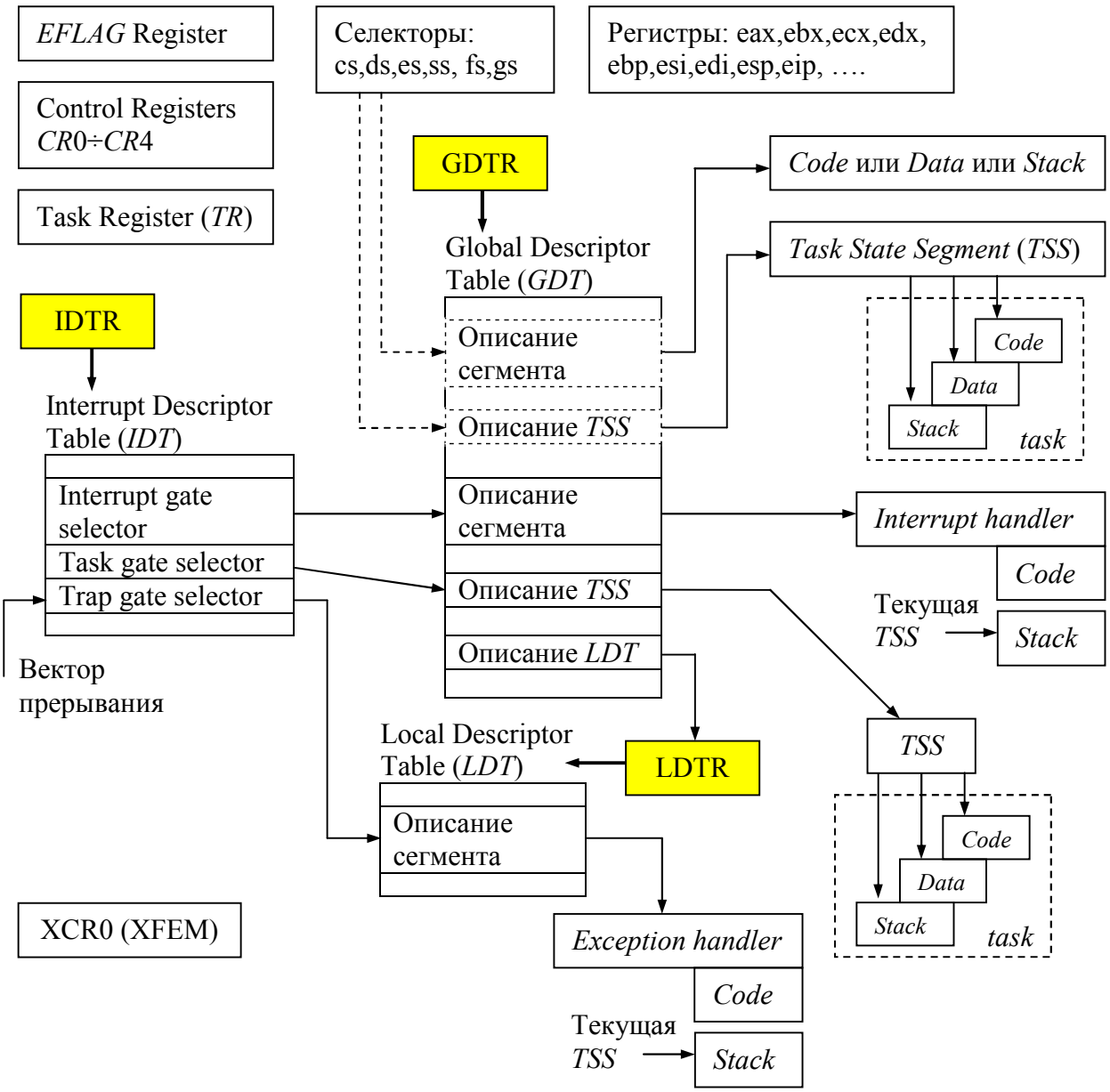


Рис. 43 – Структура процессора *IA*-32 (системный уровень)

Память подразделяется на страницы и сегменты фактически в любом сочетании и их размере. Сегмент – это блок памяти размером до 4 Гбайт (в реальном режиме сегменты имеют фиксированную длину 64 Кб). Предусмотрен аппаратный механизм выгрузки сегментов на диск и их загрузке при необходимости. Страница – это тоже блок памяти размером 4 КБ или 2 МБ или 4 МБ. При этом любая страница может отображаться на любую область физической памяти (страничное преобразование может быть отключено).

Применительно к адресации различают три адресных пространства: логическое, линейное и физическое (рис.44). Основным понятием адресации является дескриптор – это 8-байтная (64 бита) структура данных, которая определяет положение, размер блока в памяти, его назначение и характеристики защиты. Все дескрипторы хранятся в таблицах, обращение к которым поддерживается процессором аппаратно. Существуют три вида таблиц:

- глобальная (*GDT*) содержит дескрипторы, доступные всем задачам (ее размер и адрес хранятся в регистре *GDTR* процессора);
- локальная (*LDT*) может быть собственной для каждой задачи и содержит дескрипторы, описывающие элементы памяти, к которым имеет доступ только данная задача (ее размер и адрес хранятся в регистре *LDTR* процессора);
- прерываний (*IDT*) содержит описания всех прерываний процессора (ее размер и адрес хранятся в регистре *IDTR* процессора).

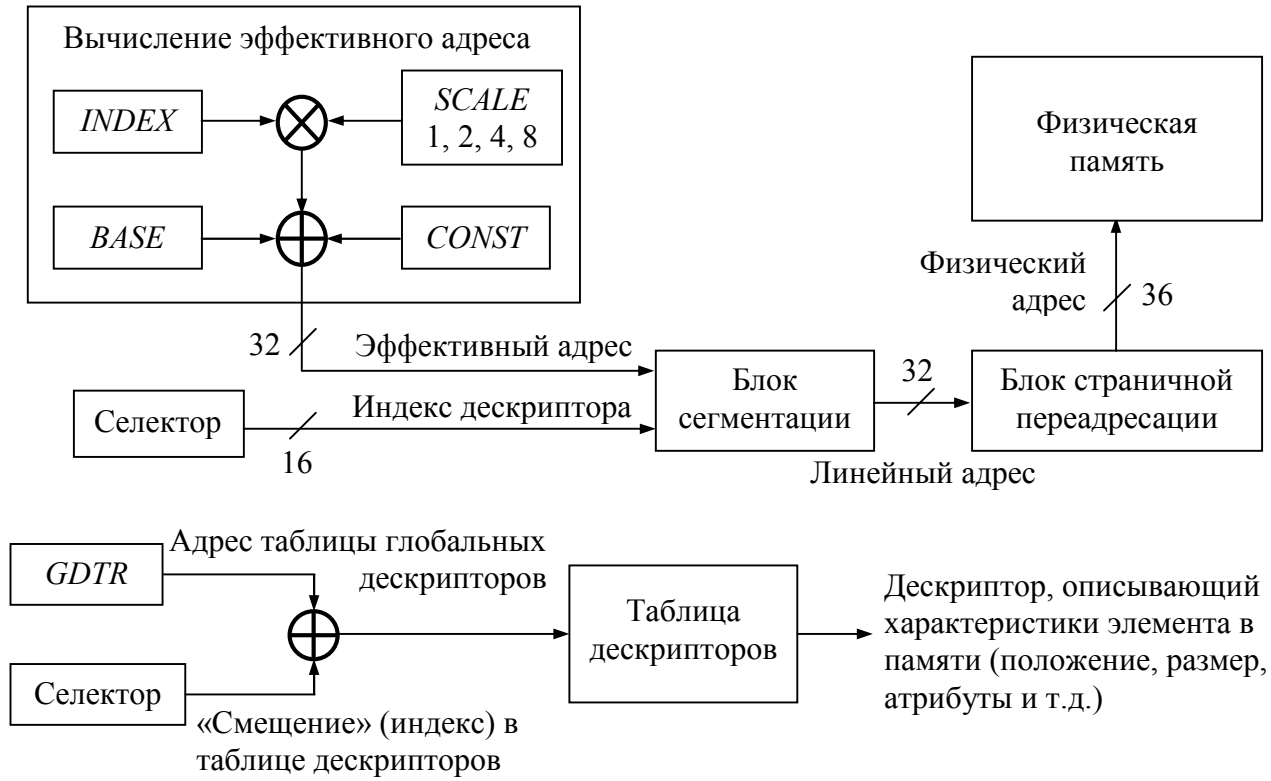


Рис. 44 – Формирование адреса памяти в защищенном режиме и механизм вычисления линейного адреса

Логический адрес (виртуальный адрес) состоит из селектора, включающего индекс (14 бит) и уровень привилегий (2 бита), и эффективного адреса (смещения). Напомним, в реальном режиме логический адрес состоит из сегмента (16 бит) и смещения (16 бит). Эффективный адрес вычисляется по формуле $EA = Base + Index \cdot Scale + Const$ (которые указываются непосредственно в команде). Поскольку каждая задача может иметь до 2^{14}=16384 селектора, а смещение ограничено размером 4 Гбайта (=2^{32}), то логическое пространство виртуальной памяти достигает 64 Тбайт. Селекторы хранятся в сегментных регистрах, а смещение в регистрах процессора или указывается непосредственно в команде.

Блок сегментации транслирует логическое пространство в 32-разрядное пространство линейных адресов. Линейный адрес образуется сложением базового адреса сегмента с эффективным адресом. В реальном режиме базовый адрес получается умножением на 16 содержимого сегментного регистра. В защищенном режиме базовый адрес загружается из дескриптора, хранящегося в таблице, по индексу дескриптора (номеру строки таблицы), который указывается в сегментном регистре (рис.44).

Физический адрес памяти (выводится на внешнюю шину процессора) образуется после преобразования линейного адреса блоком страничной переадресации. При отключенной страничной адресации линейный адрес совпадает с физическим.

В табл.12 приведены режимы адресации процессора *IA*-32.

Таблица 12. Виды адресации

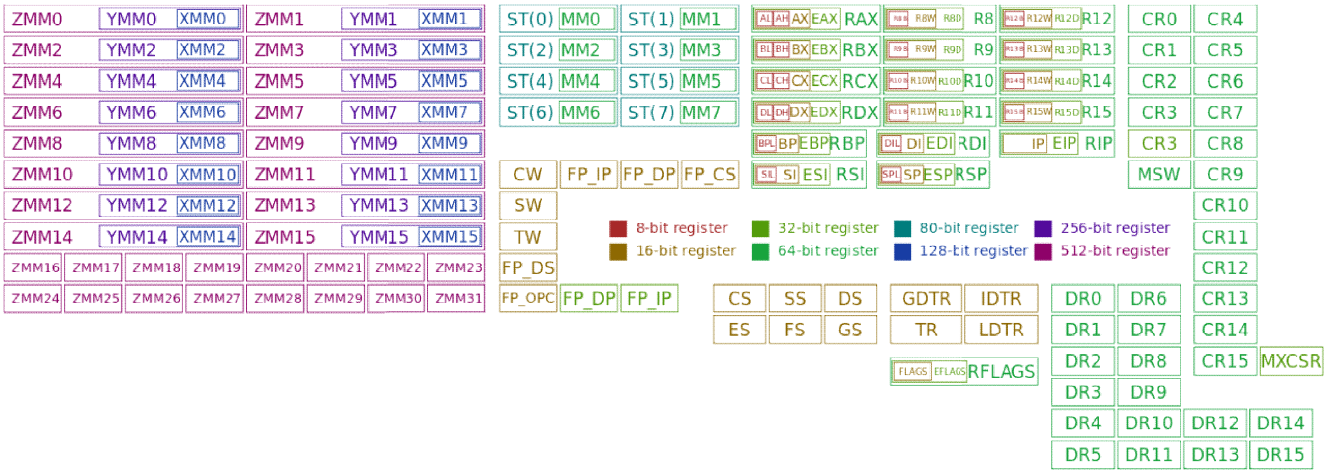
Режим	Адрес
Прямая адресация (*direct mode*), адрес памяти содержится непосредственно в команде в виде числа	*EA=Const*
Косвенная регистровая адресация (*Register indirect mode*), адрес содержится в базовом регистре	*EA=Base*
Базовая адресация (*Based mode*), адрес содержится в базовом регистре и команде в виде числа	*EA=Base+Const*
Индексная адресация (*Index mode*), адрес содержится в индексном регистре и команде в виде числа	*EA=Index+Const*

Масштабированная индексная адресация (*Scaled index mode*), адрес в индексном регистре и команде в виде числа и указания множителя	*EA=Scale*Index+Const*
Базово-индексная адресация (*Based index mode*), адрес содержится в базовом и индексном регистрах	*EA=Base+Index*
Масштабированная базово-индексная адресация (*Based scaled index mode*)	*EA= Base+ Scale*Index*
Базово-индексная адресация со смещением (*Based index mode with displacement*)	*EA= Base+Index+Const*
Масштабированная базово-индексная адресация со смещением (*Based scaled index mode with displacement*)	*EA= Base+ Scale*Index+Const*

4.1 64-битный режим

К началу 2000-х годов стало очевидно, что 32-битное адресное пространство архитектуры x86 ограничивает производительность приложений, работающих с большими объёмами данных (32-разрядное адресное пространство позволяет процессору осуществлять непосредственную адресацию лишь 4 ГБ данных). Этого может оказаться недостаточным для некоторых приложений, связанных, например, с обработкой видео или обслуживанием баз данных. Для решения этой проблемы *Intel* разработала новую архитектуру *IA*-64 – основу семейства процессоров *Itanium*. Для обеспечения обратной совместимости со старыми приложениями, использующими 32-разрядный код, в *IA*-64 был предусмотрен режим эмуляции. Однако на практике данный режим работы оказался чрезвычайно медленным.

Компания *AMD* предложила альтернативное решение проблемы увеличения разрядности процессора. Вместо того чтобы изобретать совершенно новую систему команд, было предложено ввести 64-разрядное расширение к уже существующей 32-разрядной архитектуре x86. Первоначально новая архитектура называлась x86-64, позже она была переименована в *AMD*64. Первоначально новый набор инструкций поддерживался процессорами семейств *Opteron*, *Athlon* 64 и *Turion* 64 компании *AMD*. Успех процессоров, использующих технологию *AMD*64, наряду с вялым интересом к архитектуре *IA*-64, побудили *Intel* лицензировать набор инструкций *AMD*64. При этом был добавлен ряд специфических инструкций, не присутствовавших в изначальном наборе *AMD*64. Новая версия архитектуры получила название *EM*64*T*. В рамках этой архитектуры регистры стали 64-разрядными, например: EAX → RAX.



Регистры современного процессора x86-64

Регистры для работы с памятью: адреса системных таблиц (*GDTR*, *IDTR*).

Расположение таблицы глобальных дескрипторов задается командой: LGDT (SGDT – прочитать текущее состояние), аргументом которых выступает структура включающая 32- (64) разрядный адрес местоположения (начала) таблицы глобальных дескрипторов и 16-разрядный ее размер (после сигнала сброса – в качестве начальных значений адреса находятся нули, а в качестве размера -65535). Каждый элемент таблицы глобальных дескрипторов по своей сути является структурой (8 байт =64 бита):

бит 31 16	15 бит 0
Базовый адрес (биты 0-15)	Размер сегмента (биты 0-15)

63 56	55	54	53	52	51 48	47	46 45	44	43 40	39 бит 32
Базовый адрес (биты 24-31)	G	D /B	L	A V L	Размер сегмента (16-19)	P	DPL	S	Тип сегмента	Базовый адрес (биты 16-23)

- Базовый адрес (32 бита) – начало сегмента памяти (необязательное выравнивание сегментов на границу параграфа позволяет достичь максимальной производительности механизма адресации);
- Размер сегмента (20 бит) – совместно с битом размерности G (*granularity*) задает размер сегмента (если G=0, то размер от 1 байта до 1 Мбайта, шаг 1 байт; если G=1, то от 4 Кбайт до 4 Гбайт, шаг 4 Кбайта);
- Тип сегмента и бит S – задают интерпретацию данных сегмента (код, данные, стек, системная область, права доступа, типы обращений и т.п.);
- DPL – уровень привилегий сегмента (0-максимум, 3-обычный);
- Бит P – признак присутствия сегмента в памяти (1-есть; 0-отсутствует, например: выгружен на диск);
- AVL – доступен для целей программиста;
- L – признак *native* 64-битного сегмента кода;
- Бит D/B=1 в 32-битных системах.

Расположение таблицы дескрипторов прерываний задается командой LIDT (SIDT – получить текущее состояние), аргументом которых выступает структура включающая 32- (64) разрядный адрес местоположения (начала) таблицы и 16-разрядный ее размер. Каждый элемент таблицы дескрипторов прерываний также является структурой (8 байт =64 бита).

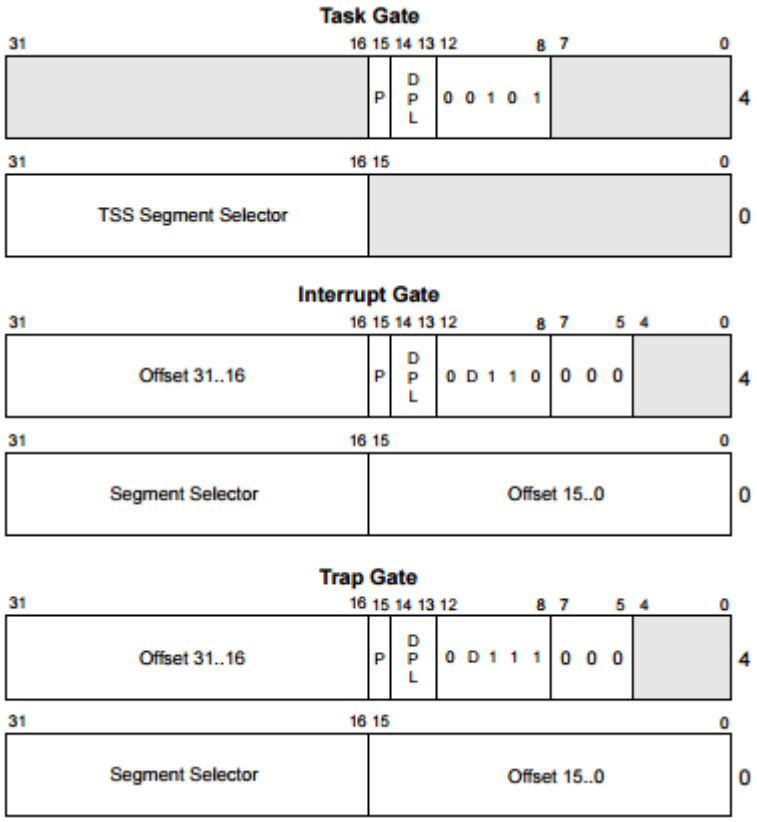
Прерывания можно разделить на три группы:

- внутренние, возникающие в самом процессоре – исключения (*exception*), номера 0-31;
- внешние, возникающие по входу *INTR* – аппаратные прерывания, подразделяются на маскируемые и немаскируемые;
- программные, вызванные командой *INT nn*.

Исключения – особые случаи подразделяются на отказы, ловушки и аварийные завершения.

1) Отказ (*fault*) – это исключение, которое обнаруживается и обслуживается до выполнения инструкции, вызывающей ошибку. После обслуживания этого исключения управление возвращается на ту же инструкцию (включая префиксы), которая вызвала отказ (пример – обращение к памяти, выгруженной на диск).
2) Ловушка (*trap*) – это исключение, которое обнаруживается и обслуживается после выполнения инструкции, его вызывающей. После обслуживания этого исключения управление возвращается на инструкцию, следующую за вызвавшей ловушку. К классу ловушек относятся и программные прерывания.
3) Аварийное завершение (*abort*) – это исключение, которое не позволяет точно установить инструкцию, его вызвавшую. Используется для сообщения о серьезных сбоях, повреждениях системных таблиц и т.п.

Таблица дескрипторов прерываний может содержать три вида дескрипторов: *task-gate*, *interrupt-gate*, *trap-gate*. На рис.45 представлены форматы этих дескрипторов.



- Offset – смещение в сегменте для точки входа в обработчик
- Segment selector – селектор сегмента с обработчиком
- D – разрядность *gate* (1 -32 бита, 0 -16 бит)
- Остальное – см. *GDTR*
- зарезервировано

Рис. 45 – Дескрипторы таблицы прерываний

На рис.46 представлена структура образования адреса обработчика исключения (*n* – количество исключений в таблице). Конкретная строка таблицы выбирается аппаратно по номеру исключения (возникшего в ходе работы системы или вызванное программистом). Далее с помощью полученного дескриптора вычисляется адрес программы обработки исключения и передается ей управление (при наличии соответствующих разрешений).

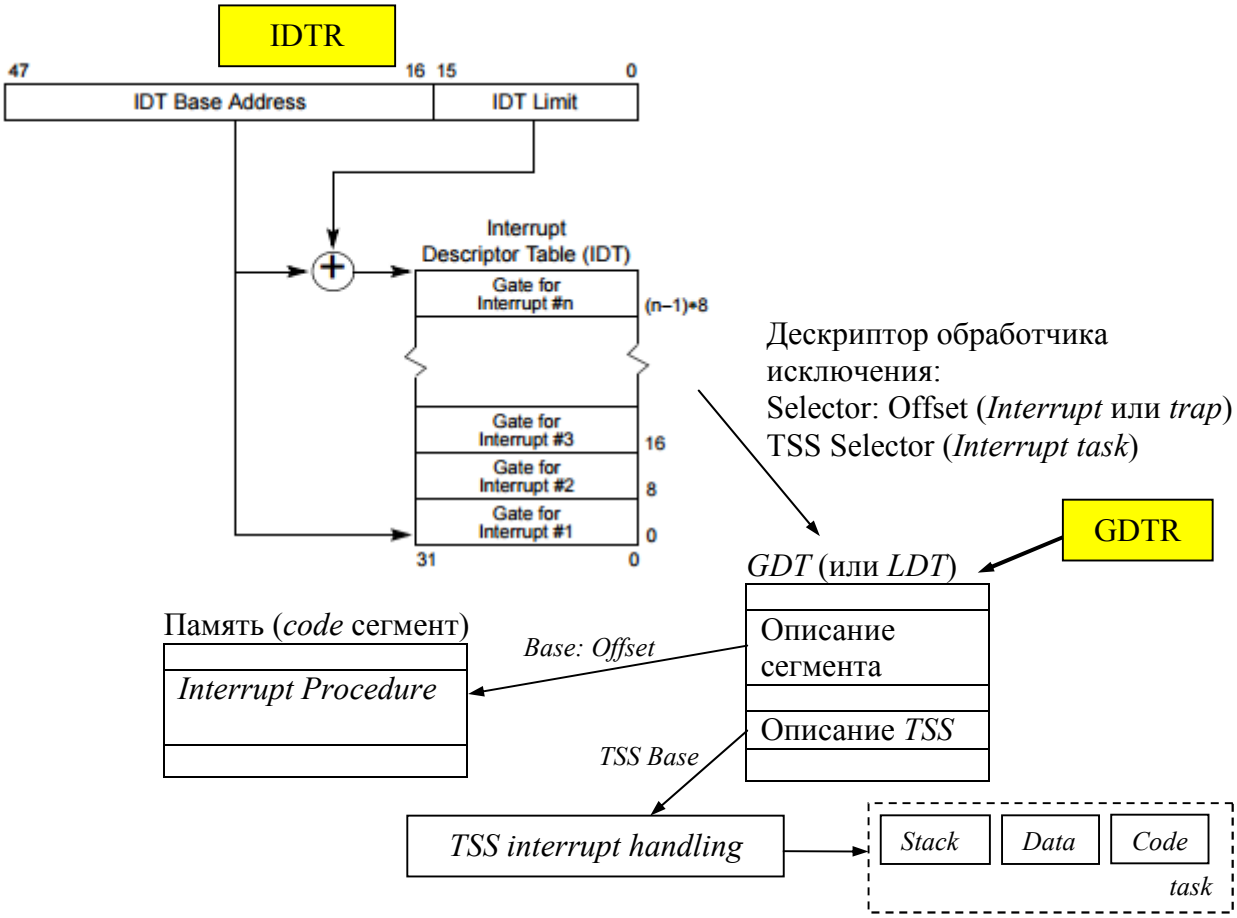


Рис. 46 – Образование адреса обработчика

При выполнении программы процессор производит анализ условий обслуживания прерываний и исключений (виды представлены в табл.13) в следующем порядке:

1. Проверка на исключение-ловушку отладки по выполненной инструкции (пошаговый режим через флаг *TF* или точка останова по данным через регистры отладки)
2. Проверка на исключение-отказ отладки по следующей инструкции (точка останова по инструкции через регистр отладки)
3. Немаскируемое прерывание (аппаратное по входу *NMI*)
4. Маскируемое прерывание при флаге *IF*=1 (аппаратное)
5. Проверка на исключение-отказ сегментации при выборке следующей команды (*NP*, *GP*)
6. Проверка на исключение-отказ страницы при выборке следующей команды (*PF*)
7. Проверка на отказ декодирования следующей инструкции (*UD*, *GP*)
8. Для операции *WAIT* проверка битов *TS* и *MP* регистра *CR*0 (исключение *NM* если они установлены в единицу)
9. Для операции *ESCAPE* (инструкция математического сопроцессора) проверка битов *TS* и *EM* регистра *CR*0 (исключение *NM* если они установлены в единицу)
10. Для операций *WAIT* или *ESCAPE* проверка на исключение *MF* от сопроцессора
11. Проверка на отказ сегментации (*NP*, *SS*, *GP*) или отказ страницы (*PF*) для операндов, используемых в инструкции.

Таблица 13. Прерывания и исключения защищенного режима

№	Функция	Мнемоника	Тип
0	Деление на ноль	*DE*	*Fault*
1	Исключение отладки	*DB*	*Fault*/*Trap*
2	Немаскируемое прерывание (NMI)	–	*Interrupt*
3	Исключение отладки (INT 3)	*BP*	*Trap*
4	Исключение по переполнению (INTO)	*OF*	*Trap*
5	Прерывание по контролю диапазона (BOUND)	*BR*	*Fault*
6	Недопустимый код операции	*UD*	*Fault*
7	Сопроцессор недоступен или переключилась задача	*NM*	*Fault*
8	Двойной отказ	*DF*	*Abort*
9	Нарушение границы сегменты сопроцессором (только 386/387)	–	*Fault*
10	Недопустимый сегмент состояния задачи	*TS*	*Fault*
11	Сегмент отсутствует	*NP*	*Fault*
12	Нарушение границы сегмента стека или сегмент стека отсутствует	*SS*	*Fault*
13	Общее нарушение защиты	*GP*	*Fault*
14	Отказ страницы	*PF*	*Fault*
15	Зарезервировано	–	–
16	Исключение сопроцессора	*MF*	*Fault*
17	Контроль выравнивания (486+)	*AC*	*Fault*
18	Машинный контроль (P5+)	*MC*	*Abort*
19-31	Зарезервировано	–	–
32-255	Аппаратные (маскируемые) и программные прерывания	–	*Trap*

5. МАТЕМАТИЧЕСКИЙ СОПРОЦЕССОР

Математический сопроцессор (*FPU – floating point unit*) предназначен для расширения возможностей центрального процессора и позволяет проводить вычисления с плавающей точкой, а также вычислять значения некоторых функций (логарифм, степень числа, тригонометрические функции и т.п.).

Сопроцессор поддерживает семь типов данных: 16-, 32-, 64- битные целые числа; 32-, 64-, 80- битные числа с плавающей точкой и 18-и разрядные числа в двоично-десятичном формате. Формат чисел соответствует стандартам *IEEE* 754, 854. Т.е. все действительные числа хранятся в нормализованном виде (число у которого целая часть состоит из одной не равной нулю цифры): мантисса и порядок. В двоичной системе эта целая часть всегда равна 1 и ее не хранят, что позволяет сэкономить 1 бит мантиссы. В поле порядка записывается степень числа 2, на которую умножается мантисса, плюс смещение, равное 16383 для расширенной точности, 1023 – для двойной точности и 127 – для одинарной точности. Рассмотрим представление числа 7.375 при одинарной точности. Заметим, что 7.375=4+2+1+1/4+1/8. Поэтому в двоичной системе это число следует записать в виде: $0111.011=2^2 1.11011$. Таким образом мантисса: 11011, а порядок равен 2+127=129 (10000001). Поэтому положительное число 7.375 при одинарной точности будет храниться в виде:

(бит 31) (бит 0)

0 10000001 11011000000000000000000

Сопроцессор работает параллельно с ЦП, выполняя только свои специфические команды. Программная модель сопроцессора состоит из восьми 80-и разрядных регистров, организованных по принципу стека, 16-и разрядных регистров статуса и управления, слова тэгов (16 бит) и указателей инструкции и данных.

У каждого регистра данных *ST*(0)-*ST*(7) имеется два бита (тэг), которые объединены в слово тэгов. Значение этих битов позволяет быстро анализировать состояние регистра:

00 (*Valid*) – указывает на наличие числа в регистре

01 (*Zero*) – нулевое значение регистра

10 (*Special*) – специальное назначение

11 (*Empty*) – пустой

Слово состояния (*status word*) отображает общее состояние сопроцессора

15			12				8	7			4				0
B	*C*3	*TOP*			*C*2	*C*1	*C*0	*ES*	*SF*	*PE*	*UE*	*OE*	*ZE*	*DE*	*IE*

B=1 – (*Busy*) сопроцессор занят выполнением операции или имеется не обслуженный запрос на прерывание

TOP – указывает на вершину стека

*C*3-*C*0 – определяют код условия (*condition code*)

ES – (*Error Summary Status*) устанавливается при возникновении немаскируемого исключения

SF – (*Stack Flag*) устанавливается при некорректной операции со стеком (переполнение сверху или снизу)

Биты 0-5 устанавливаются при возникновении соответствующих исключений

Управляющее слово (*Control word*) служит для выбора опций выполняемых операций

15			12				8	7			4				0
x	x	x	x	*RC*		*PC*		x	x	*PM*	*UM*	*OM*	*ZM*	*DM*	*IM*

RC (*Round Control*) – определяет способ округления результата:

00 – к ближайшему значению

01 – по направлению к минус бесконечности

10 – по направлению к плюс бесконечности

11 – по направлению к нулю

PC (*Precision Control*) – задает точность вычислений и тип операндов

00 – одинарная точность (24 бита)

01 – зарезервировано
10 – двойная точность (53 бита)
11 – расширенная (64 бита)

Единичные значения в битах 0-5 маскируют соответствующие исключения

Указатели инструкций и данных служат для сохранения физического адреса, кода выполняемой операции и физического адреса операнда.

Таблица 14. Исключения сопроцессора

Флаг	Маска	Исключение
IE	*IM*	*Invalid Operation* – недействительная операция: извлечение данных их пустого регистра; неопределенный результат (0/0, беск/беск); извлечение корня из отрицательного числа и т.п.
DE	*DM*	*Denormalized Operand* – денормализован хотя бы один операнд
ZE	*ZM*	*Zero Divide* – деление на ноль ненулевого операнда
OE	*OM*	*Overflow* – переполнение сверху
UE	*UM*	*Underflow* – переполнение снизу
PE	*PM*	*Precision* – результат не может быть точно представлен в заданном формате, при этом выполняется округление

5.1 ОСНОВНЫЕ КОМАНДЫ СОПРОЦЕССОРА

Арифметические команды

FABS – нахождения модуля числа: st(0)=ABS(st(0))
FCHS – изменение знака: st(0)=–st(0)

FADD mem – сложение вещественных чисел: st(0)=st(0)+[mem]
st(i),st(0): st(i)=st(i)+st(0)
st(0),st(i): st(0)=st(0)+st(i)
FADDP – аналогична FADD, но с извлечением из стека

FSUB mem – вычитание вещественных чисел: st(0)=st(0)–[mem]
st(i),st(0): st(i)=st(i)–st(0)
st(0),st(i): st(0)=st(0)–st(i)
FSUBR – обратное вычитание (аналогично FSUB, но меняются местами вычитаемое и уменьшаемое)
FSUBP, FSUBRP – аналогичны командам FSUB и FSUBR, но с извлечением из стека

FMUL mem – умножение вещественных чисел: st(0)=st(0)*[mem]
st(i),st(0): st(i)=st(i)*st(0)
st(0),st(i): st(0)=st(0)*st(i)
FMULP – аналогична FMUL, но с извлечением из стека

FDIV mem – деление вещественных чисел: st(0)=st(0)/mem
st(i),st(0): st(i)=st(i)/st(0)
st(0),st(i): st(0)=st(0)/st(i)
FDIVR – обратное деление (аналогично FDIV, но меняются местами делитель и делимое)
FDIVP, FDIVRP – аналогичны командам FDIV и FDIVR, но с извлечением из стека

FSQRT – извлечение квадратного корня из числа st(0)
FSCALE – умножение числа st(0) на округленную в сторону нуля степень числа 2, хранящуюся в st(1)
FRNDINT – округление числа st(0) до целого в соответствии с флагами RC слова управления

(*) команды FIADD, FISUB, FIMUL, FIDIV и их эквиваленты с суффиксами R и/или P работают с целочисленными операндами

Команды сравнения

FCOM mem – сравнение вещественных чисел:
установка флагов по результату операции st(0)–[mem]
st(i): st(0)–st(i)

FCOMP – аналогична FCOM, но с извлечением из стека

FCOMPP – сравнение чисел st(0) и st(1), двойное извлечение из стека

FICOM, FICOMP, FICOMPP – аналогичны командам FCOM, FCOMP, FCOMPP но работают с целочисленными операндами

FUCOM, FUCOMP, FUCOMPP – аналогичны командам FCOM, FCOMP, FCOMPP но без генерации исключения NaN результата

FTST – команда сравнения st(0) с нулем и установки по результату флагов

Биты флагов *FPU*			Результат
*C*3	*C*2	*C*0	
0	0	0	st(0) больше операнда
0	0	1	st(0) меньше операнда
1	0	0	st(0) равно операнду
1	1	1	st(0) не сравнимо с операндом

Команды загрузки операндов и записи результата

FLD mem – загрузка вещественной константы в регистр st(0)
st(i): st(0)=st(i)

FST mem – запись вещественного числа st(0) в память
st(i): st(i)=st(0)

FSTP – аналогична FST, но с извлечением

FILD, FIST, FISTP – аналогичные командам FLD, FST, FSTP, но работают с целочисленными операндами

FLD1 – загрузка в st(0) единицы: st(0)=+1.0

FLDZ – st(0)=0.0

FLDPI – st(0)=pi

FLDL2E – st(0)=$\log_2(e)$

FLDL2T – st(0)= $\log_2(10)$

FLDLG2 – st(0)=lg(2)

FLDLN2 – st(0)=ln(2)

Другие команды

FFREE st(i) – освобождение регистра st(i) (сброс его тега)

FINIT – инициализация FPU

FSTCW / FLDCW – запись/загрузка слова управления FPU. В качестве операнда может быть регистр AX или память

FSTSW / FLDSW – запись/загрузка слова состояния FPU. В качестве операнда может быть регистр AX или память

5.2 ПРИМЕР РАБОТЫ С СОПРОЦЕССОРОМ

Рассмотрим задачу вычисления корней квадратного уравнения: $ax^2 + bx + c = 0$.

Алгоритм решения

1) находим дискриминант: $D^2 = b^2 - 4ac$
2) если $D^2 \geq 0$, то имеется два действительных корня

$x_1 = st(0) = \frac{-b + D}{2a}$, $x_2 = st(1) = \frac{-b - D}{2a}$, регистр BX=1

3) если $D^2 < 0$, то имеются комплексно-сопряженные корни

$st(0)=\frac{-b}{2a}$ – действительная часть корней

$st(1)=\frac{\sqrt{|D^2|}}{2a}$ – модуль мнимой части корней

регистр BX=2

Положим, что коэффициенты уравнения хранятся в памяти

```
;
mov bx,0
fld qword ptr ds:[b]
fmul st(0),st(0);              возвели коэффициент b в квадрат
fld qword ptr ds:[a]
fmul qword ptr ds:[c]
fimul word ptr ds:[c_4];              умножение на константу 4
fsubp st(1),st(0);             аналогичная команда fsubp
ftst;          сравниваем квадрат дискриминанта с нулем
fstsw ax ; результат сравнения отражается на флагах сопроцессора
fabs
fsqrt
fld qword ptr ds:[a]
fadd st(0),st(0);     умножение коэффициента a на константу 2
test ah,5; проверка битов C2 и C0
jz L_real; переход если квадрат дискриминанта неотрицателен
; иначе - комплексные корни
; st(0)=2a             st(1)=sqrt(|D^2|)
fdiv st(1),st(0);     st(1)=st(1)/st(0)
fdivr qword ptr ds:[b];     st(0)=b/st(0)
fchs
; в регистрах st(0), st(1) находятся действительная и мнимая части комплесно-
; сопряженных корней
mov bx,2
jmp L_
;
L_real:
; st(0)=2a             st(1)=sqrt(|D^2|)
fld qword ptr ds:[b]
fchs
fsub st(0),st(2);     st(0)=–b–D
fdiv st(0),st(1);     st(0)=(–b–D)/2a
fld qword ptr ds:[b]
fchs
fadd st(0),st(2);     st(0)=–b+D
fdiv st(0),st(1);     st(0)=(–b+D)/2a
ffree st(3);  удалили значение дискриминанта
ffree st(2);  удалили значение 2a
; в регистрах st(0), st(1) находятся действительные корни уравнения
mov bx,1
jmp L_
;
L_:
…..
```

5.3 ТЕХНОЛОГИЯ *MMX*

Технология *MMX* ориентирована на приложения мультимедиа и коммуникации. Это расширение базовой архитектуры процессоров серии i80x86 появилось вместе со вторым поколением *Pentium*. *MMX* состоит в одновременной обработке нескольких элементов данных за одну инструкцию. Эта концепция в более общем виде известна как технология *SIMD* (*Single instruction – Multiple data*).

Физически технология *MMX* реализована на сопроцессоре с некоторыми особенностями. Регистры *MMX*(0)-*MMX*(7) размещены в регистрах *FPU*, но их адресация абсолютная. Любая инструкция *MMX* обнуляет поле *TOP* регистра состояния *FPU* и обнуляет биты тэга соответствующего регистра *FPU*.

Еще одной особенностью технологии *MMX* является поддержка арифметики с насыщением (ее отличие от обычной арифметики с циклическим переполнением заключается в том, что при возникновении переполнения в результате фиксируется максимально большое число, а перенос игнорируется).

Расширение *MMX* использует новые типы упакованных 64-битных целочисленных данных и добавляет 57 инструкций для одновременной обработки нескольких элементов данных. Это арифметические, логические инструкции, инструкции сравнения, сдвига, пересылки и очистки.

Четыре типа данных (выступают операндами команд *MMX*):
- восемь упакованных байт (8 x8 бит)
- четыре упакованных слова (4 x16 бит)
- два упакованных двойных слова (2 x32 бита)
- учетверенное слово (64 бита)

Все команды *MMX* имеют формат: *instruction dest, src* (*instruction* – мнемоника команды, *dest* – операнд для результата и он же «исходные данные 1», *src* – операнд «исходных данных 2»). Большинство команд имеют суффикс, который определяет тип данных и используемую арифметику:

- US (*unsigned saturation*) – арифметика с насыщением, данные без знака.
- S или SS (*signed saturation*) – арифметика с насыщением, данные со знаком.
- Если в суффиксе нет ни S, ни US, используется циклическая арифметика (*wraparound*).
- B, W, D, Q указывают тип данных (байты -8 бит, слова -16 бит, двойные слова -32 бита, учетверенные слова -64 бита). Если в суффиксе есть две из этих букв, первая соответствует входному операнду, а вторая – выходному.

Команды *MMX* исполняются в том же режиме процессора, что и команды с плавающей запятой. Поэтому при их исполнении (кроме команды EMMS) "портится" слово состояния регистров с плавающей запятой. В ознакомительных целях ниже приводятся доступные команды (обозначения: mm – *MMX*-регистр; m32, m64 – память объема 32 и 64 бит соответственно; imm – непосредственный операнд; ir32 – целочисленный регистр МП):

EMMS	Команда обеспечивает переход процессора от исполнения *MMX*-команд к исполнению обычных команд с плавающей запятой: она устанавливает значение 1 во всех разрядах слова состояния.
PADDB mm, mm/m64 PADDW mm, mm/m64 PADDD mm, mm/m64	Команды складывают элементы данных (байты, слова или двойные слова) входного и выходного операнда. Если сумма выходит за границу допустимого диапазона, то по правилам циклической арифметики избыток отсчитывается от другой границы диапазона. "Переноса" единицы из одного элемента данных в другой не происходит.
PADDSB mm, mm/m64 PADDSW mm, mm/m64 PADDUSB mm, mm/m64 PADDUSW mm, mm/m64	Команды складывают элементы данных (байты или слова) входного и выходного операнда. Если сумма выходит за граничное значение допустимого диапазона, то результатом считается это граничное значение.
PSUBB mm, mm/m64 PSUBW mm, mm/m64 PSUBD mm, mm/m64	Команды вычитают элементы данных (байты, слова или двойные слова) входного операнда из элементов данных выходного операнда. Если результат выходит за границу допустимого диапазона, то по правилам циклической арифметики соответствующее число единиц отсчитывается от другой границы диапазона. "Переноса" единицы из одного элемента

	данных в другой не происходит.
PSUBSB mm, mm/m64 PSUBSW mm, mm/m64 PSUBUSB mm, mm/m64 PSUBUSW mm, mm/m64	Команды вычитают элементы данных (байты или слова) входного операнда из элементов данных выходного операнда. Если разность выходит за граничное значение допустимого диапазона, то результатом считается это граничное значение.
PSLLW mm, mm/m64/imm PSLLD mm, mm/m64/imm PSLLQ mm, mm/m64/imm	Команды выполняют сдвиг влево элементов данных (16-, 32- или 64-разрядных слов) в выходном операнде на число бит, задаваемое входным операндом. Освободившиеся младшие разряды заполняются нулями.
PSRLW mm, mm/m64/imm; PSRLD mm, mm/m64/imm; PSRLQ mm, mm/m64/imm	Команды выполняют сдвиг вправо элементов данных (16-, 32- или 64-разрядных слов) в выходном операнде на число бит, задаваемое входным операндом. Освободившиеся старшие разряды заполняются нулями.
PSRAW mm, mm/m64/imm PSRAD mm, mm/m64/imm	Команды выполняют сдвиг вправо элементов данных (16- или 32-разрядных слов) в выходном операнде на число бит, задаваемое входным операндом. Если сдвигается положительное число, то освободившиеся старшие разряды заполняются нулями, а если отрицательное, то единицами.
PAND mm, mm/m64	Команда вычисляет поразрядное логическое «И» своих операндов.
PANDN mm, mm/m64	Команда вычисляет поразрядное отрицание выходного операнда, а затем поразрядное логическое «И» между входным операндом и обращенным значением выходного.
POR mm, mm/m64	Команда вычисляет поразрядное логическое «ИЛИ» своих операндов.
PXOR mm, mm/m64	Команда вычисляет поразрядное логическое «исключающее ИЛИ» своих операндов.
PMADDWD mm, mm/m64	Команда попарно перемножает 16-разрядные слова со знаком входного и выходного операндов. Это дает четыре 32-разрядных произведения. Затем первое произведение складывается со вторым, а третье с четвертым. Суммы записываются в 32-разрядные слова выходного операнда. Если все слова на входе равны 8000h, результатом будет 80000000h (это единственный случай, когда перемножение отрицательных чисел дает отрицательный результат).
PMULHW mm, mm/m64	Команда попарно перемножает 16-разрядные слова со знаком входного и выходного операндов, что дает четыре 32-разрядных произведения. Старшие разряды произведений записываются в 16-разрядные слова выходного операнда. Младшие разряды произведений теряются.
PMULLW mm, mm/m64	Команда попарно перемножает 16-разрядные слова со знаком входного и выходного операндов, что дает четыре 32-разрядных произведения. Младшие разряды произведений записываются в 16-разрядные слова выходного операнда. Старшие разряды произведений теряются.
PCMPEQB mm, mm/m64 PCMPEQW mm, mm/m64 PCMPEQD mm, mm/m64	Команды попарно сравнивают элементы данных (байты, 16- или 32-разрядные слова) входного и выходного операндов. Если элемент данных выходного операнда равен соответствующему элементу входного, такой элемент выходного операнда заполняется единицами. Если равенства нет, он заполняется нулями.
PCMPGTB mm, mm/m64 PCMPGTW mm, mm/m64 PCMPGTD mm, mm/m64	Команды попарно сравнивают элементы данных (байты, 16- или 32-разрядные слова со знаком) входного и выходного операндов. Если элемент данных выходного операнда больше соответствующего элемента входного, такой элемент выходного операнда заполняется единицами; если же он не больше входного, то он заполняется нулями.
PACKSSWB mm, mm/m64 PACKSSDW mm, mm/m64	Команды преобразуют длинные элементы данных (16- и 32-разрядные слова со знаком) в более короткие (байты или 16-разрядные слова со знаком). Если исходное значение было за пределами допустимого диапазона для выходного типа данных, то результатом упаковки считается ближайшее граничное значение диапазона.
PACKUSWB mm, mm/m64	Команда преобразует 16-разрядные слова со знаком из обоих операндов в байты без знака и записывает их в выходной операнд. Если исходное слово со знаком было больше FFh, результатом преобразования

	считается FFh. Если исходное слово со знаком отрицательно, результатом преобразования считается 00h.
PUNPCKHBW mm, mm/m64 PUNPCKHWD mm, mm/m64 PUNPCKHDQ mm, mm/m64	Команды попарно объединяют исходные элементы данных (байты, 16- или 32-разрядные слова), находившиеся в старших 32 разрядах обоих операндов. Полученные в результате более длинные элементы данных записываются в выходной операнд. Исходные значения младших разрядов операндов на результат не влияют.
PUNPCKLBW mm, mm/m64 PUNPCKLWD mm, mm/m64 PUNPCKLDQ mm, mm/m64	Команды попарно объединяют исходные элементы данных (байты, 16- или 32-разрядные слова), находившиеся в младших 32 разрядах обоих операндов. Полученные в результате более длинные элементы данных записываются в выходной операнд. Исходные значения старших разрядов операндов на результат не влияют.
MOVD mm, mm/m32/ir32	Команда копирует 32 бита из младших разрядов *MMX*-регистра, либо из памяти, либо из целочисленного регистра в младшие 32 разряда *MMX*-регистра (старшие разряды заполняются нулями).
MOVD m32/ir32, mm	Команда копирует 32 бита из младших разрядов *MMX*-регистра в память либо в целочисленный регистр.
MOVQ mm, mm/m64	Команда пересылки данных в *MMX*-регистр.
MOVQ mm/m64, mm	Команда пересылки данных из *MMX*-регистра.

6. Развитие архитектуры *Intel* x86 процессоров

Совершенствование процессора i8086 связано с увеличением быстродействия (производительности), возможностями подключения большего объема памяти, большего количества внешних устройств, снижения энергопотребления. Это достигается следующими факторами:

1) совершенствование архитектуры:
   - буферизация (кэширование) данных и команд;
   - переход от шинно-мостовой к хабовой архитектуре, а затем к модульной структуре (распределение потоков и децентрализация);
   - построение ядра *CISC* процессора на *RISC*-подобной архитектуре;
   - размещение однотипных ядер на одном кристалле;
   - интеграция периферии процессора на кристалл;
   - дублирование структур процессора (в рамках одного ядра);
   - конвейеризация исполнения команд;
   - переход от параллельного выполнения команд к параллельному выполнению микроопераций и их кэшированию;
   - развитие алгоритмов предсказания переходов;
2) технологические улучшения:
   - уменьшение технологических норм производства;
   - применение новых материалов;
   - применение новых принципов построения транзистора (объемных);
3) расширение возможностей:
   - внедрение новых интерфейсов (протоколов) для связи с «внешним миром»;
   - совершенствование шин адреса, данных, управления;
   - внедрение многопроцессорности и работы с несколькими потоками команд.

Расположив технические усовершенствования в хронологическом порядке, можно отследить тип задач и способы их решения, которые сопровождали развитие семейства x86 процессоров.

i8086 F_{clk}=4-10 МГц	16-разр. архитектура, адресация до 1 Мбайт памяти (20 бит), внешний сопроцессор i8087
i80286 F_{clk}=6-24 МГц	16-разр. архитектура, адресация до 16 Мбайт памяти (24 бита), внешний улучшенный сопроцессор i80287
i80386 F_{clk}=16-40 МГц	32-разр. архитектура, адресация до 4 Гбайт памяти (32 бита), механизм виртуализации памяти (до 64 Гб), внешний сопроцессор i80387, интеграция периферии в одну БИС 82360, функции энергосбережения, строковые команды (*ins*, *outs*, *movsd*, *stosd*, *lodsd*, *scasd*)
i80486 F_{clk}=16-120 МГц *FSB*: 16-55 МГц	32-разр. архитектура (полная совместимость с i8086) ▪ в процессор введен кэш первого уровня (8 Кбайт), представляющий собой ассоциативную структуру единую для данных и инструкций, кэш второго уровня достигал 512 Кбайт (внешний); ▪ повышена производительность локальной шины процессора за счет введения пакетных циклов, позволяющие передавать данные в каждом такте процессора, а не через такт как прежде; ▪ введены буферы отложенной записи, позволяющие процессору решать проблему временной занятости шины, и, не останавливаясь, продолжать выполнение программы; ▪ в процессоре применено *RISC* ядро, позволяющее наиболее часто встречающиеся инструкции выполнять за один цикл; ▪ процессор имеет 5-ступенчатый конвейер; ▪ математический процессор интегрирован на кристалл; ▪ впервые (в 1992 году) применена технология умножения частоты шины (когда частота процессора в *k* раз больше частоты шины *front side bus*);

	▪ введены дополнительные команды (bswap, xadd, cmpxchg и т.д.).
Pentium (*P*5) *P54C* F_{clk}=60-200 МГц *FSB*: 50-66 МГц	По базовой регистровой архитектуре и системе команд процессоры являются 32-х разрядными, но имеют 64-х разрядную шину данных, основные достижения: ▪ суперскалярная архитектура (*UV*-архитектура): процессор имеет два параллельно работающих конвейера и два АЛУ, благодаря чему можно выполнять две инструкции одновременно (не всякие инструкции), причем большинство команд выполняются за 1 цикл; ▪ технология динамического предсказания переходов (глубина 256 шт.) совместно с внутреннем кэшем команд объемом 8 Кбайт обеспечивает максимальную загрузку конвейеров (впервые кэш разделен на две части – для команд и данных); ▪ внутренний кэш данных объемом 8 Кбайт работает с отложенной до освобождения шины записью и настраивается на режим сквозной или обратной записи; ▪ введены новые инструкции; ▪ более быстрый аппаратный умножитель в несколько раз сокращает (и делает более предсказуемым) время выполнения инструкций MUL и IMUL по сравнению с 80486, время выполнения уменьшается с 13-42 тактов до 10-11 для 32-битных операндов; ▪ улучшена архитектура и производительность сопроцессора: некоторые инструкции ускорились на порядок, например скорость выполнения FMUL увеличилась в 15 раз, инструкция FXCH ST(x) работает параллельно с обычными инструкциями (арифметическими или загрузки/выгрузки регистров); ▪ реализован механизм поддержки двухпроцессорной системы; ▪ в процессоре реализованы средства отладки.
Pentium Pro (*P*6) F_{clk} до 200 МГц	Началом 6-го поколения процессоров принято считать выпуск этого процессора (1995 год). Идеи и технологии, заложенные в данный чип, определили архитектуры всех современных x86 процессоров: блоки предсказания ветвлений, переименование регистров, *RISC*-ядро, интегрированная в один корпус с ядром кэш-память второго уровня. Но главным отличием было применение технологии динамического исполнения (изменения порядка исполнения инструкций) и двойной независимой шины, благодаря чему сняты ограничения на пропускную способность памяти. Однако технологическая сложность ядра данного процессора привела к сравнительно невысокому выходу годных чипов при технологиях того времени, что сказалось на высокой цене. При этом процессор обладал достаточно низкой производительностью при исполнении 16-разрядного кода (ОС были 16-разр).
Pentium MMX *P*55*C* F_{clk}=133-300 МГц *FSB*: 60, 66 МГц	На «закате» 5-го поколения процессоров (1997) было выпущено расширение системы команд *MMX* (*multimedia extension*), которое было ориентировано на приложения мультимедиа и коммуникации (по сути это 57 новых инструкций, призванных ускорить обработку видео и звука). Основная идея «технологии *MMX*» заключается в одновременной обработке нескольких элементов данных за одну инструкцию – *SIMD* (*Single instruction – Multiple data*) и реализовывалась на базе сопроцессора.
Pentium II (*P*6) F_{clk} до 450 МГц *Pentium* III (*P*6) F_{clk} до 1.4 ГГц *FSB*: до 133 МГц	В это поколение (имеющее обобщенное обозначение *P*6) также входят процессоры *Celeron* («упрощенные» версии МП: меньшие тактовые частоты, урезанный кэш и т.п.) и *Pentium M* (доработанный *Pentium* III для мобильных платформ). Главным отличием процессоров *P*6 от предыдущего поколения является динамическое исполнение программы, состоящее в том, что инструкции процессором могут выполняться не в том порядке, в котором

	они следуют в программе. Но при этом все внешние операции записи, чтения будут выполняться в том порядке как они обозначены в программе. Это решение призвано повысить производительность (фактически впервые – не за счет увеличения тактовой частоты или разрядности шины). Кроме того, процессоры *P*6 имеют двойную независимую шину: одна шина (системная) служит для общения ядра с основной памятью и интерфейсными устройствами, а другая – предназначена исключительно для обмена с кэшем второго уровня. Также в систему команд процессора введены специальные команды условной пересылки данных, позволяющие сократить число условных переходов и повысить предсказуемость кода, т.е. увеличить эффективность загрузки конвейера. В процессорах реализованы функции поддержки многопроцессорных систем (до 2-х *CPU*). С выпуском *Pentium* III впервые представлено расширение *SSE* (*Streaming SIMD Extensions*) – потоковое *SIMD* расширение, которое представляет собой набор дополнительных инструкций, поддерживающий вычисления с плавающей точкой (со скалярными и упакованными типами данных). *SSE* состоит из восьми 128-битных регистров (с *xmm*0 до *xmm*7). Каждый регистр определяет 4 последовательных значения с плавающей точкой одинарной точности (по 32 бита).
Pentium 4 (P7) (*NetBurst, Prescott*), *Pentium D* (*Smithfield*= 2 x *Prescott*) F_{clk}=1.4-3.8 ГГц *FSB* (*QDR*): 400, 533, 800 МГц	Это принципиально новый процессор (2000 год) с гиперконвейеризацией (*hyper-pipelining*) – конвейером, состоящим из 20 и более ступеней и с применением кэша последовательностей микроопераций вместо традиционного кэша инструкций, также АЛУ процессоров архитектуры *NetBurst* имеет существенные отличия от архитектур других процессоров. Согласно заявлениям *Intel*, процессоры, основанные на данной технологии, позволяют добиться увеличения производительности примерно на 40% относительно семейства *P*6 при одинаковом технологическом процессе и при «правильной» загрузке процессора. На практике же, первые образцы процессоров работали даже медленнее, чем *Pentium* III. С появлением *Pentium* 4 было анонсировано *SSE*2, а также технология *Hyper-threading* (позволяющая ОС видеть вдвое больше ядер – технология виртуальной многоядерности). *SSE*2 – улучшенное расширение *SSE*, которое производит потоковые вычисления с вещественными числами двойной точности (2 числа по 64 бита в одном регистре *SSE*). Кроме того, добавлены инструкции, аналогичные расширению *MMX*, работающие с регистрами *SSE* (16 байт, 8 слов, 4 двойных слова или 2 учетверённых слова в регистре). *SSE*2 включает в себя ряд команд управления кэшем. В 2004 году с обновлением ядра процессора (переход на *Prescott*) добавлено *SSE*3, которое в частности оптимизирует технологию *Hyper-threading*, также появляется поддержка технологии *EM*64*T* – *Extended Memory* 64 *Technology* (в ранних процессорах поддержка 64-битных расширений была отключена). Расширение *SSE*3 содержит 13 новых инструкций, наиболее заметное изменение – возможность горизонтальной работы с регистрами (добавлены команды сложения и вычитания нескольких значений, хранящихся в одном регистре). Эти команды упростили ряд *DSP* и 3D-операций. Существует также новая команда для преобразования значений с плавающей точкой в целые без необходимости вносить изменения в глобальном режиме округления. Надо сказать, что микроархитектуры *NetBurst*, *Prescott* и их усовершенствованные клоны не оправдали ожиданий (фактически процессоры *Pentium* 4 ознаменовали исчерпание возможностей архитектуры заложенной с первого *Pentium*): ▪ увеличение разрядности шин (параллельной), увеличения их частоты

	(*quad data rate – QDR*, физически частота шины в 4 раза ниже); ▪ роста множителя частоты процессора; ▪ увеличение числа ступеней конвейера процессора; ▪ дублирование устройств процессора для возможности одновременного выполнения некоторых команд; ▪ расширение системы команд в соответствии с потребностями (*MMX*, *SSE*) и для повышения предсказуемости кода при условных и безусловных переходах в программе; ▪ выполнения ядра по *RISC* микроархитектуре; и т.д. Ключевыми недостатками процессоров семейств *Pentium* 4 и *Pentium D* с точки зрения конечного потребителя являлись относительно невысокая производительность, высокое энергопотребление и запредельное тепловыделение.
Core /*Core* 2 *Duo*|*Quad* F_{clk} до 3.3|3.0 ГГц *FSB* (*QDR*): 1066, 1333, 1600 МГц *Core* 2 *Conroe* (65 нм): *conroe* (*duo*) *merom* (*duo*) *allendale* (*duo*) *kentsfield* (*quad*) (=2x *conroe*) *Core* 2 *Penryn* (45 нм): *wolfdale (duo)* *yorkfield* (*quad*) (=2x *wolfdale*)	Поколение процессоров *Core* /*Core* 2 – это некий возврат назад с применением современных (на 2006 год) технологий. *Intel*, понимая сложившуюся ситуацию с поколением *Pentium* 4, приняла решение обратиться к другой ветви свой продукции – в основе новых процессоров лежит переработанное ядро *Pentium M*, которое представлялось «панацеей от всех бед», нажитых *Intel* в течение предшествующих шести лет. Таким образом, ядро *P*6, использованное ещё в процессорах *Pentium Pro* (1995 год), продолжило свою эволюцию: ▪ выпускается по новому техпроцессу (рост частоты с 0.15 до 3 ГГц); ▪ появилась новая системная шина; ▪ поддержка многоядерности (на уровне кристалла); ▪ повышение эффективности ядра и улучшение кэша; ▪ расширение мультимедийных инструкций; ▪ переход от погони за максимальной производительностью к оптимальному соотношению быстродействия и энергопотребления. Именно с этих позиций инженеры создавали новую архитектуру *Core Microarchitecture*, которая легла в основу процессоров: 1) *Core* – для ноутбуков, одно- и двухъядерное (исполняющее 32-бит. код) 2) *Core* 2 – как в настольном (*Conroe*), так и мобильном исполнении (*Merom*), включают ряд микроархитектурных улучшений и способны исполнять 64-битный код (*EM*64*T*), количество ядер: 1, 2 или 4. Особенностями процессоров *Core* 2 являются: ▪ поддержка *Enhanced Memory* 64 *Technology* (*EM*64*T*); ▪ *Intel Virtualization Technology* – аппаратная виртуализация (технология поддержки виртуальных x86-машин *Vanderpool*); ▪ *NX*-бит – бит запрета исполнения, добавленный в страницы для реализации возможности предотвращения выполнения данных как кода; ▪ *Intel Advanced Digital Media Boost* – переработка блоков исполнения *SIMD* инструкций (*SSE*, *SSE*2, *SSE*3) – современное ПО (для обработки изображений, видео и звука, шифрования, научной и финансовой деятельности) достаточно широко использует наборы команд *SSE*, которые позволяют работать со 128-битовыми операндами различного характера (векторами и целочисленными либо вещественными данными повышенной точности), что заставило *Intel* задуматься об ускорении работы *SSE* блоков процессора (до этого момента МП исполняли одну *SSE* инструкцию, работающую с 128-битными операндами, лишь за два такта: один такт тратился на обработку старших 64 бит, второй такт – на обработку младших и теперь *SSE* инструкций исполняются за 1 такт), т.е. блоки *SSE* в будущих процессорах будут полностью 128-битовыми, что даёт возможность увеличить количество данных, обрабатываемых процессором за такт;

- набор инструкций *SSSE*3 (*Supplemental Streaming SIMD Extension* 3) – 16 уникальных команд, работающих с упакованными целыми, каждая из которых может работать как с 64-х битными (*MMX*), так и с 128-ми битными *xmm* регистрами;
- многоядерный дизайн на одном кристалле (1,2,4 ядра);
- кэш-память первого уровня объёмом 64 Кбайта, которая разделена на две части по 32 Кбайта для кода и данных, что более подходит для *RISC* ядра процессора;
- *Intel Advanced Smart Cache* – общая, разделяемая на ядра, кэш-память *L*2 объёмом 2 или 4 Мб – во-первых, у процессора появляется возможность гибко регулировать размеры областей кэша, используемых каждым из ядер (на примере двух ядер: доступ ко всему объёму *L*2 кэша может получить любое из ядер: когда одно бездействует, второе получает в своё распоряжение весь объём кэш-памяти, если же одновременно работают два ядра, то кэш делится между ними пропорционально, в зависимости от частоты обращений каждого ядра к ОЗУ, более того, если оба ядра работают синхронно с одними и теми же данными, то хранятся они в общем *L*2 кэше только однократно); во-вторых объединённая *L*2 кэш-память позволяет значительно снизить нагрузку на ОЗУ и на процессорную шину (в этом случае перед системой не стоит задача контроля и обеспечения когерентности кэш-памяти различных ядер, в то время как в системах с двухъядерными процессорами с раздельными кэшами, в случае, если оба ядра работают с одними и теми же данными, эти данные дублируются в кэш-памяти каждого из ядер – возникает необходимость в контроле их актуальности, т.е. перед тем, как извлечь такие данные из *L*2 кэша для обработки, каждое процессорное ядро должно проверить, не изменило ли эти данные другое ядро – требуется обновление содержимого кэш-памяти, которое в системах на базе МП с микро-архитектурой *NetBurst* выполняется через системную шину и ОЗУ).
- *Intel Wide Dynamic Execution* – динамическое исполнении команд (от *Pentium Pro*), обладающее возможностями упреждающего и внеочередного исполнения команд (в каждое ядро добавлено дополнительные декодер и исполнительные устройства);
- *Intel Smart Memory Access* – технологии, объединенные под этим собирательным названием, направлены на уменьшение задержек, которые могут возникнуть при доступе процессора к обрабатываемым данным (для этой цели подходит предварительная выборка данных из памяти в обладающие гораздо более низкой латентностью *L*1 и *L*2 кэши процессора, отметим, что алгоритмы предварительной выборки данных эксплуатируются в процессорах *Intel* достаточно давно, но с выходом микроархитектуры *Core* соответствующий функциональный узел усовершенствован):
  - микроархитектура содержит ≥6 независимых блоков предварительной выборки данных (два блока нагружаются задачей предварительной выборки данных из памяти в общий *L*2 кэш, ещё по два блока работают с кэшами *L*1 каждого из ядер *CPU*), каждый из этих блоков независимо друг от друга отслеживает закономерные обращения (потоковые, либо с постоянным шагом внутри массива) исполнительных устройств к данным и, основываясь на собранной статистике, блоки предварительной выборки стремятся подгружать данные из памяти в процессорный кэш ещё до того, как к ним последует обращение; также, *L*1 кэш каждого из ядер *CPU* имеет по одному блоку предварительной выборки инструкций, работающий по аналогичному принципу;

	- технология *memory disambiguation* (устранение противоречий в памяти), которая направлена на повышение эффективности работы алгоритмов внеочередного исполнения инструкций, осуществляющих чтение и запись данных в памяти (в *CPU*, осуществляющих внеочередное исполнение команд, не допускается выполнение команды чтения до того, как не будут завершены все инструкции сохранения данных, что зачастую снижает загрузку исполнительных устройств и эффективность работы *CPU* в целом, для решения этой проблемы технология предусматривает специальные алгоритмы, позволяющие с достаточно высокой вероятностью устанавливать зависимость последовательных команд сохранения и загрузки данных, и даёт возможность, таким образом, применять внеочередное выполнение инструкций к таким командам (в случае ошибки в определении зависимых инструкций загрузки и сохранения данных, которые, согласно информации разработчиков «случаются достаточно редко», технология детектирует возникший конфликт, перезагружает корректные данные и инициирует повторное исполнение "ошибочно" выполненного кода); ▪ выполнение каждым ядром до 4-х инструкций за такт; ▪ длина конвейера 14 стадий; ▪ более совершенный блок предсказания переходов; ▪ более вместительные буферы команд, используемые на различных этапах анализа кода для оптимизации скорости исполнения; ▪ *LaGrande Technology* – аппаратная технология защиты информации; ▪ *EIST* (*Enhanced Intel SpeedStep Technology*) усовершенствованная технология *SpeedStep*; ▪ *Intel Intelligent Power Capability* – возможность отключения «на лету» тех собственных подсистем, которые не используются в данный момент (речь в данном случае идёт не о ядрах целиком), что обеспечивается декомпозицией *CPU* на отдельные функциональные узлы на низком уровне (каждое ядро поделено на большое количество блоков и внутренних шин, питание которыми осуществляется раздельно), при этом работа технологии не влечёт за собой увеличение времени отклика *CPU* на внешние воздействия, вызванное необходимостью приводить отключенные блоки в функциональное состояние; ▪ *iAMT*2 (*Active Management Technology*) – удаленное управление компьютерами; ▪ новый набор команд *SSE*4.1, впервые реализованный в процессорах *Penryn* (*SSE*4 состоит из 54 инструкций, 47 из них относят к *SSE*4.1, ни одна из *SSE*4 инструкций не работает с 64-битными *MMX* регистрами, только со 128-битными *xmm*0-15); добавленные инструкции включают: ускорение компенсации движения в видеокодеках, быстрое чтение из *USWC* памяти, множество инструкций для упрощения векторизации программ компиляторами (работа с векторными примитивами, скалярное умножение векторов), впервые в *SSE*4 регистр *xmm*0 стал использоваться как неявный аргумент для некоторых инструкций. Основными недостатками архитектуры является: все запросы к памяти проходят через северный мост (хотя *Intel* попыталось это скомпенсировать кэшем *L*2), не поддерживается технология *Hyper-threading*, также блок *FPU* относительно слаб по отношению к конкурентам.
Nehalem *Core i*7	Развитие «идей» *Core* 2, позволило в 2008 г. создать модульную структуру, позволяющую варьировать количество ядер, со встроенным контроллером памяти (3- или 2- канальной *DDR*3) и новой шиной *QPI* (т.к. классическая

Gulftown (32нм) F_{clk}=3.2-3.5 ГГц *Bloomfield* (45нм) F_{clk}=2.6-3.33 ГГц *Lynnfield* (45нм) F_{clk}=2.8-3.1 ГГц *Arrandale* (32нм) *Core i*5 *Lynnfield* (45нм) *Clarkdale* (32нм) *Arrandale* (32нм) *Core i*3 *Clarkdale* (32нм) *Arrandale* (32нм) *Intel Pentium* *Arrandale* (32нм) *Clarkdale* (32нм) *Intel Celeron* *Arrandale* (32нм) *Clarkdale* (32нм) *Jasper Forest* (32нм) шина *QPI* (*quick path interconnect*)	*FSB* исчерпала свои «возможности»), соединяющую *CPU* с чипсетом. Улучшения микроархитектуры приводят к большей производительности у *Core i*7 в сравнении с *Core* 2 на равных частотах. Позже (2009-10 года) появились *Core i*5/*i*7 с двухканальным контроллером памяти и 4-я ядрами, затем – *Core i*3/*i*5 с двумя ядрами и встроенным видеоядром (мобильный сегмент). Среди производительных решений выпускаются *Core i*7 с трехканальным контроллером памяти и 6/8-ю ядрами (благодаря технологии *Hyper-threading* эти *CPU* могут исполнять до 12/16 потоков команд соответственно). *CPU Nehalem* с 4-я ядрами содержат не менее 731 млн. транзисторов, что на 10% меньше, чем у процессоров *Yorkfield* (*Core* 2 *quad*), но площадь кристалла увеличилась с 214 до 263 мм². Первые *CPU Nehalem* основаны на том же 45-нм техпроцессе, что и *Penryn*. Наиболее значимые изменения: ▪ встроенный контроллер памяти, поддерживающий 2 или 3 канала *DDR*3, *SDRAM* или 4 канала *FB-DIMM*; ▪ новая шина *QPI* (до 25.6 Гбайт/сек), пришедшая на смену классической шине *FSB* (только в *CPU* для *LGA*1366; *CPU* для *LGA*1156 используют шину *DMI* (связь с южным мостом), т.к. северный мост встроен в процессор); ▪ возможность выпуска процессоров со встроенным *GPU* (в бюджетных решениях на базе 2-х ядерных *CPU*); ▪ в отличие от *Kentsfield* и *Yorkfield* (*Core* 2 *quad Conroe* и *Penryn* соответственно), которые состоят из двух кристаллов по 2 ядра в каждом, все 4 ядра *Bloomfield* находятся на одном кристалле; ▪ добавлен кэш 3-го уровня; ▪ поддержка *SMT* (организация 2-х логических ядер из одного физического – наследие *hyper-threading*); ▪ *SSE*4.2 (7 команд) – инструкции обработки строк 8/16 битных символов, вычисления *CRC*32, подсчета популяции единичных бит.
Sandy Bridge (32нм) F_{clk}=2.8-3.6 ГГц шина *QPI* Процессоры этой архитектуры (как и *Nehalem*) известны на рынке под торговыми марками *Core i*7 /*Core i*5 /*Core i*3 /*Pentium* /*Celeron*	Микроархитектура *CPU* анонсирована в 2011 г., *Intel* планировала перевести на нее процессоры всех ценовых сегментов. Технически задача состояла так: сделать *CPU* на той же площади кристалла и том же 32 нм техпроцессе, который на старых программах имел бы +10% скорости и -20% потребления одновременно, а на новых (где используются добавленные команды) желательно удвоение пиковой скорости при том же *TDP*. Первые изделия – это *CPU* с частотой до 3.5 ГГц с 2 или 4 ядрами и *GPU* с частотой до 1.35 ГГц (*Intel HD Graphics* 2000, для *K* серии – *HD Graphics* 3000). Также в *CPU* интегрирован северный мост набора системной логики (контроллер *PCI Express* 2.0 и двухканальный контроллер памяти стандарта *DDR*3 *SDRAM* с частотой до 1333 МГц). Каждое ядро имеет по 256 Кб *L*2 кэша и объединенный кэш *L*3 до 8 Мб. Процессор, графика, кэш-память и контроллеры выполнены на единой кремниевой подложке площадью 216 мм². Потребление не выходит за пределы 130 Вт для топовых моделей. Новая микроархитектура несёт поддержку новых *SIMD* инструкций для работы с векторными вычислениями *AVX* (*Advanced Vector Extensions*), которые дополняют *SSE*. Новый набор, оставаясь обратно совместимым, увеличивает разрядность 128-разрядных регистров *xmm*0-*xmm*15 до 256 бит (*ymm*0-*ymm*15), а также даёт в распоряжение дополнительные трёх- и четырёхоперандные команды (было заявлено, что использование *AVX* способно поднять скорость работы некоторых алгоритмов до 90%). Основные нововведения: ▪ первая архитектура с технологией *Quick Sync*, которая предназначена

	для ускорения кодирования и декодирования видеоконтента (в составе графического ядра сделаны специализированные аппаратные модули); ▪ поддерживаются технологии *Advanced Encryption Standard* (*AES*) и *Virtualization Machine Extensions* (*VMX*); ▪ добавлен кэш инструкций нулевого уровня (*L*0), содержащий до 1536 декодированных микроопераций для экономии энергии и улучшения пропускной способности инструкций (блок предварительной выборки может отключать декодер инструкций, если обнаруживает в кэше уже декодированную инструкцию); ▪ появился абсолютно новый блок переупорядочивания команд, основанный на использовании физического регистрового файла; ▪ оптимизированы и улучшены алгоритмы предсказателя переходов; ▪ введен блок *DRM* под названием «*Intel Insider*», который является «дополнительным уровнем защиты контента»; ▪ функция *vPro* – возможность удаленного управления, например удаленного блокирования ПК или стирания информации с НЖМД. Структуру чипа *Sandy Bridge* можно условно разделить на четыре основных элемента: процессорные ядра, графическое ядро, кэш-память *L*3, «Системный агент» (*System Agent*). Эти элементы объединены с помощью 256-битной межкомпонентной кольцевой шиной, выполненной на основе новой версии технологии *QPI*. Шина состоит из четырёх колец: ▪ шины данных (*Data Ring*) – 32-байта +*ECC*; ▪ шины запросов (*Request Ring*); ▪ шины мониторинга состояния (*Snoop Ring*); ▪ шины подтверждения (*Acknowledge Ring*). Основные преимущества кольцевой топологии шины: ▪ высокая масштабируемость (до 20 ядер на кристалл); ▪ снижение задержки кэша *L*3, и перевод его на частоту процессора; ▪ использование графическим ядром кэша *L*3. Производительность кольцевой шины достигает 96 Гбайт/сек при тактовой частоте 3 ГГц, что фактически в четыре раза превышает показатели процессоров *Intel* предыдущего поколения. Улучшения позволили увеличить удельную производительность *Sandy Bridge* по сравнению с процессорами поколения *Nehalem* почти на 15%. К этому добавился рост номинальных тактовых частот и отличный разгонный потенциал.
Ivy Bridge (22 нм) F_{clk}=3.4, 3.5 ГГц	Несмотря на то, что применялась новая производственная норма 22 нм, тактовые частоты *CPU* совсем не выросли. Сделанные улучшения в дизайне в основном коснулись контроллера памяти и контроллера шины *PCI Express*, который получил совместимость с третьей версией данного стандарта. Микроархитектура вычислительных ядер претерпела лишь отдельные косметические переделки, которые позволили ускорить выполнение операций деления и небольшого увеличения эффективности технологии *Hyper-Threading*. В результате, рост производительности составил не более 5 процентов. Начиная с процессоров этого поколения, *Intel* отказалась от сопряжения полупроводникового кристалла *CPU* и закрывающей его крышки посредством бесфлюсовой пайки и перешла на заполнение пространства между ними полимерным термоинтерфейсным материалом с не очень хорошими теплопроводящими свойствами. Это искусственно ухудшило частотный потенциал и сделало процессоры *Ivy Bridge*, как и всех их последователей заметно менее разгоняемыми.
Haswell (22 нм) F_{clk}=3.5-4.1 ГГц	Появление поколения *Haswell*, которое, в отличие от *Ivy Bridge*, относится к фазе «так» не принесло ожидаемого роста производительности *CPU*.

	Однако различных улучшений в микроархитектуре сделано немало, причём они рассредоточены по разным частям исполнительного конвейера, что в сумме должно было бы увеличить общий темп исполнения команд (благодаря этому *Haswell* стал способен обрабатывать до восьми микроопераций за такт – на треть больше предшественников, более того, новая микроархитектура удвоила и пропускную способность кэш-памяти первого и второго уровней). Однако на практике прирост удельной производительности у *Haswell* по сравнению с *Ivy Bridge* составил лишь 5-10%. Но отметим, что на векторных операциях ускорение заметно больше, наибольший выигрыш можно увидеть в приложениях, использующих новые *AVX*2 и *FMA*-команды, поддержка которых также появилась в этой микроархитектуре: ▪ *AVX*2 это развитие *AVX* – целочисленные команды *SSE* работающие с 256-битными *AVX* регистрами; ▪ *FMA* (*Fused Multiply-Add*) – это набор 128- и 256-битных *SIMD* инструкций, предназначенный для выполнения операции умножения-сложения над числами в формате с плавающей запятой с однократным округлением. Более поздние модификации *Haswell*, получившие кодовое наименование *Devil's Canyon*, смогли нарастить преимущество над предшественниками за счет увеличения тактовой частоты (превысила 4-гигагерцовый порог). Кроме того, *Intel* улучшила полимерный термоинтерфейс под процессорной крышкой.
Broadwell (14 нм) F_{clk} до 3.8 ГГц	Основной ключевой особенностью этих *CPU* стала новая 14 нм норма производства и при этом никаких значительных нововведений в микроархитектуре не планировалось (это должен был быть обычный «тик»). Действительно успех новинки вполне мог обеспечить один только техпроцесс с *FinFET*-транзисторами второго поколения, который сулил уменьшение энергопотребления и увеличение частот. Однако внедрение новой технологии обернулось чередой неудач, в результате которых *Broadwell* досталась лишь экономичность. В итоге процессоры этого поколения (для настольных систем) вышли больше похожими на мобильные *CPU*. Кроме урезанных тепловых пакетов и снижения частот они отличаются уменьшившимся в объёме *L*3-кэшем, что, несколько компенсируется появлением расположенного на отдельном кристалле кэша четвёртого уровня (*eDRAM*). В результате на одинаковой с *Haswell* частоте процессоры *Broadwell* демонстрируют преимущество примерно в 7%. Микроархитектура имеет следующие особенности: ▪ увеличение объема внутренних буферов; ▪ уменьшение количества тактов при операциях умножения (с 5 до 3-х); ▪ увеличение ширины блока деления до 10 бит; ▪ появление *AVX* инструкций для работы с числами произвольной точности; ▪ в 1.5 раза вырос объем таблицы ассоциативной трансляции адресов второго уровня (*L*2 *TLB*); ▪ улучшение механизма предсказаний сложных ветвлений кода; ▪ усовершенствованное управление питанием.
Skylake-S (14 нм) F_{clk} до 4.3 ГГц	Это поколение представляется не столько как дальнейшее развитие микроархитектуры, сколько «работа над ошибками». В результате номинальные частоты процессоров вернулись к показателям, которые были свойственны их 22 нм предшественникам. В *Skylake* конвертер питания *CPU* вновь перекочевал на материнскую плату и снизил тем самым суммарное тепловыделение. Несмотря на то, что *Skylake*, как и *Haswell*, относится к фазе «так», нововведений в микроархитектуре совсем

	немного: ▪ улучшение результативности предсказания ветвлений ; ▪ повышение эффективности блока предварительной выборки; ▪ усовершенствована подсистема памяти, контроллер памяти получил поддержку работающей на высоких частотах памяти стандарта *DDR*4; ▪ внутрипроцессорная кольцевая шина стала быстрее, что расширило полосу пропускания *L*3-кэша; ▪ улучшения в графическом ядре и в энергоэффективности. Поэтому сопоставляя *Skylake* напрямую с *Haswell* (минуя *Broadwell*), увидим рост удельной производительности порядка 7-8%.
Kaby lake (14+ нм) *Coffee lake* (14++) – «новый степинг *Kaby lake*» F_{clk} до 4.3 ГГц	Этап оптимизации *Skylake.* *Turbo boost*: +(200-400) МГц по тактовым частотам, более агрессивная политика. Частота «встроенной графики»: +50 МГц. Производительность: плюс 10%.
Cannon lake (10 нм) F_{clk}=2.2 ГГц (3.2)	Целочисленное деление (DIV/IDIV) традиционно реализовано с помощью микрокода. Деление может занять до 90 тактов для *Skylake*. *Cannon Lake* получил улучшенный микрокод, благодаря чему процесс деления сократился до 10–18 тактов. Быстрее исполняются *AES*-инструкции (в 2.5-3 раза), что обеспечивается дополнительными *AES*-блоками. В *Cannon Lake* несколько увеличилась латентность инструкций, но при этом значительно возросла скорость их исполнения (т.е. процессор немного дольше готовится к исполнению, зато выполняет её быстрее). Также сообщается об уменьшении количества необходимых тактов для выполнения некоторых других инструкций.

6.1 Выводы

На пути от *Sandy Bridge* до *Skylake* компания *Intel* сменила две полупроводниковых технологии и уменьшила толщину транзисторных затворов более чем вдвое. Однако на 2018 год 14 нм техпроцесс по сравнению с 32 нм технологией (2010-11 года) так и не позволил нарастить рабочие частоты процессоров *Core* (последние пять поколений имеют похожие тактовые частоты, которые незначительно превышают 4 ГГц). С точки зрения удельной производительности одного ядра, то в среднем за четыре поколения (*Ivy Bridge – Skylake*) прибавка составила 14%, а самый большой скачок произошел при смене архитектуры *Ivy Bridge* с памятью стандарта *DDR*3 на *Haswell-E* с *DDR*4 – рис.47а, а с позиций отработки 14 нм техпроцесса добавили 5-10% производительности «в среднем». Что касается полезности технологии *Hyper-Threading*, то в подавляющем большинстве тестов от нее есть очевидные плюсы, поскольку при ее использовании скорость увеличивается на 18-20%. Конечно, она не в состоянии имитировать полноценное второе ядро процессора, которое, к слову, обеспечивает до 2-х кратной прибавки в производительности.

Также, по данным программных пакетов: *PCMark* 2010, *3DMark Time Spy*, *Corona* 1.3, *Blender* 2.79*b*, *V-Ray* 3.57.01, *WinRar* 5.5, кодеков x264 и x265, *VeraCrypt* 1.22, *Stockfish* 9 и др., прогресс усреднённого быстродействия в ресурсоёмких приложениях у процессоров *Core i*7 (работающих на одинаковой тактовой частоте) представлен на рис.47б (хорошо вида корреляция с данными рис.47а). Фактически *Core i*7, несмотря на малый рост тактовой частоты, с каждым новым поколением становятся лучше, что обеспечивается за счёт структурных изменений и оптимизаций в микроархитектуре. Если судить по быстродействию в приложениях для создания и обработки цифрового контента, то можно видеть, что средний прирост удельной производительности на каждом этапе составляет порядка 15 процентов.

Микроархитектура *Cannon Lake* способна обеспечить исполнение большего числа инструкций за такт (*IPC*) по сравнению с микроархитектурой *Skylake*. Прирост составляет примерно 2-6 %.

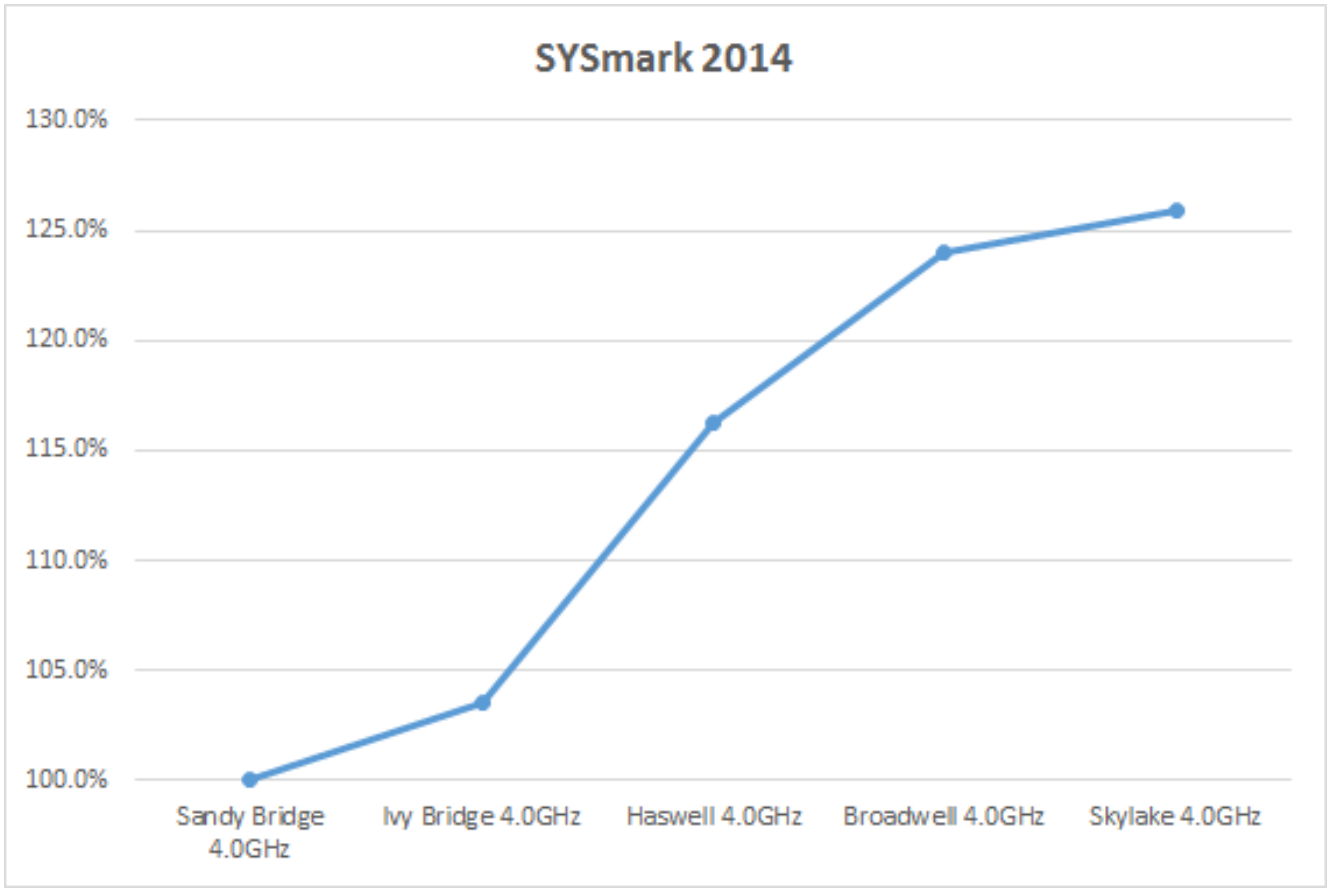


Рис. 47а – Рост производительности в семействе x86 МП за 5 поколений

Рис. 47б – Производительность процессоров x86 за 2010-2018 г.

7. Интерфейсы

7.1 Шина *QPI*

Шина *QPI* (*Quick Path Interconnect*) является аналогом шины *HyperTransport* от *AMD*, и предназначена для связи процессора с другими компонентами: связывает процессор с северным мостом и другими процессорами – рис.48. В общем случае, задачей шины *QPI* является обмен данными между небольшими группами локальных процессоров, а также взаимодействие между банками памяти (даже не обязательно одного типа) в распределенных системах, включающих не более 128 процессоров. *QPI* обеспечивает меньшие задержки и более высокую производительность, по сравнению с *HyperTransport*.

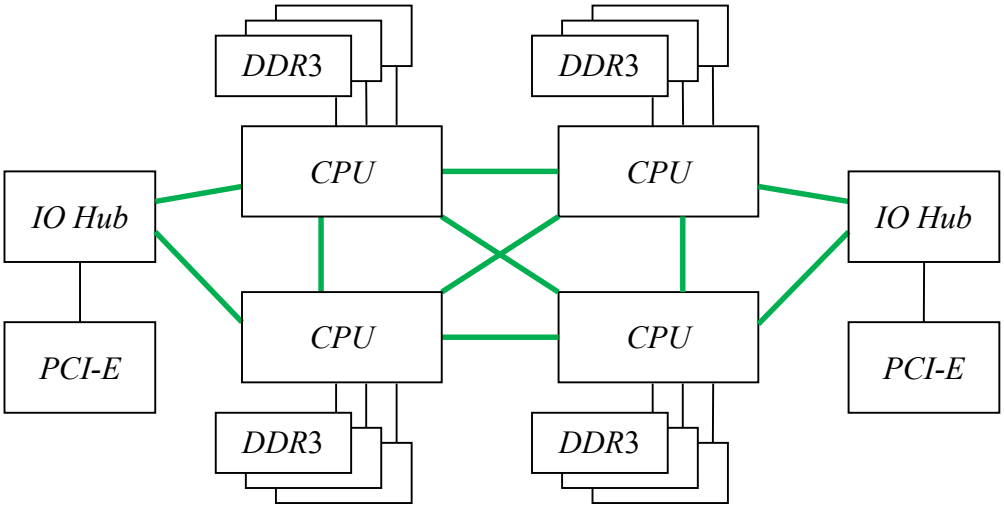


Рис. 48 – Применение *QPI* в многопроцессорных системах

Шина *QPI* для тактирования использует базовую частоту системы (в ПК обычно 133 МГц), умножая её на свой собственный коэффициент: от 4x до 64x. Например: частота шины *QPI* на разных моделях *CPU* разная: у *Core i*7-940-2.93 ГГц и *i*7-920-2.66 ГГц её частота 2.4 ГГц (множитель 18); у *Core i*7-965 *Extreme Edition* эта шина работает на частоте 3.2 ГГц (множитель 24), что эквивалентно пропускной способности 4.8 гигатранзакций в секунду (ГТ/с) и 6.4 ГТ/с соответственно (т.е. до 25.6 Гбайт/сек). Для сравнения классическая шина *FSB* 800 МГц обеспечивала 6.4 Гбайт/сек. Повышение эффективности шины обеспечивается динамическим управлением частотой и напряжением принимающего и передающего чипов, а также другими нововведениями. Кроме того, был разработан чип-диспетчер, который аппаратно распределяет каналы между ядрами процессора. В результате, производительность симулированного 64-ядерного процессора при его помощи удалось повысить в два раза. Это позволяет производить эффективные и экономичные многоядерные процессоры, т.к. технологии приёма/передачи данных требуют не только повышенной пропускной способности канала, но и более эффективного с точки зрения потребляемой мощности интерфейса передачи информации.

Особенностью архитектуры с шиной *QPI* является применение концепции масштабируемой разделяемой памяти: каждый *CPU* имеет собственную выделенную память, к которой он будет обращаться напрямую (через свой интегрированный контроллер памяти). В случае, если процессору потребуется доступ к выделенной памяти другого *CPU*, он может связаться с ней посредством одного из каналов *QPI*. Как и шина *HyperTransport*, *QPI* использует последовательную связь по схеме "точка-точка", что обеспечивает высокую скорость при малой латентности. Основными характеристиками *QPI* являются:

- производительность каналов до 6.4 гигатранзакций в секунду (благодаря чему общая пропускная способность может достигать 25.6 Гбайт/сек);
- малый объем служебной информации, необходимой для функционирования МП систем (позволяет повысить скорость передачи полезных данных);

- реализация контроля передачи данных при помощи циклического избыточного кода (*CRC*) и повторной передачи при обнаружении ошибок на канальном уровне (позволяет обеспечить целостность данных без ощутимого влияния на производительность);
- возможность реализации высокоуровневых функций обеспечения надежности, готовности и удобства обслуживания, реконфигурации каналов в случае повреждения отдельных участков (поддержке "горячей замены"); при нарушении сигнала в одной или нескольких линий контроллер шины может автоматически перенастроить *QPI* на ширину 15 и даже 5 линий, не теряя работоспособности; при организации шины с различной шириной, управление потоком данных осуществляет специальный агент, который распределяет поток данных перед тем, как отправить его по различным физическим линиям, а при приеме аналогичный агент собирает разные потоки данных в один (рис. 49).

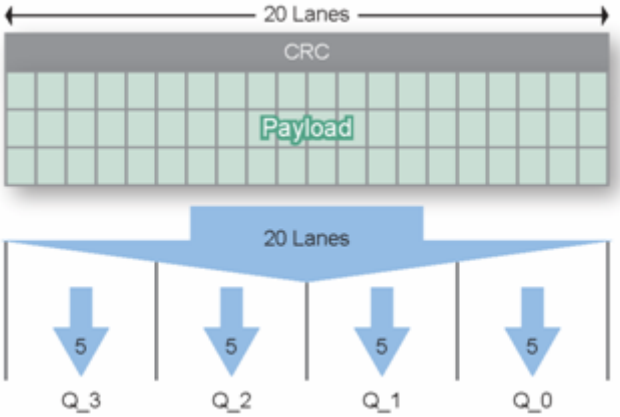


Рис. 49 – Пример конфигурирования 20 каналов в четыре группы по 5 каналов

- гибкость интерфейса – имеющиеся линии могут распределяться в соответствии с потребностями процессора при этом интерфейс не обременен доступом к собственной памяти (этим занимается его контроллер памяти) – рис.50.

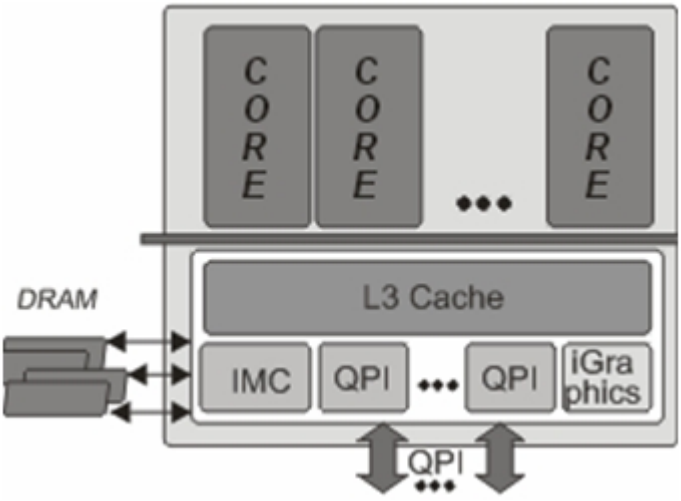


Рис. 50 – Архитектурные особенности процессоров *Core i*7 с шиной *QPI*

Специальный последовательный интерфейс с топологией точка-точка, именованный как *Quick Path Interconnect*, с технической точки зрения представляет собой два 20-битных соединения, ориентированных на одновременную передачу данных в прямом и обратном направлении – рис.51 (полный дуплекс). Таким образом, *QPI* является последовательной, высокоскоростной двунаправленной шиной (по 20 отдельных пар линий на прием и передачу), при этом 16 бит отводится для данных, две линии зарезервированы для служебных сигналов и еще две – для передачи кодов коррекции ошибок *CRC*. С учетом еще двух пар линий, используемых для сигналов синхронизации (одна на прием и одна на передачу), получается, что шина *QPI* состоит из 42 пар линий, то есть является 84-контактной.

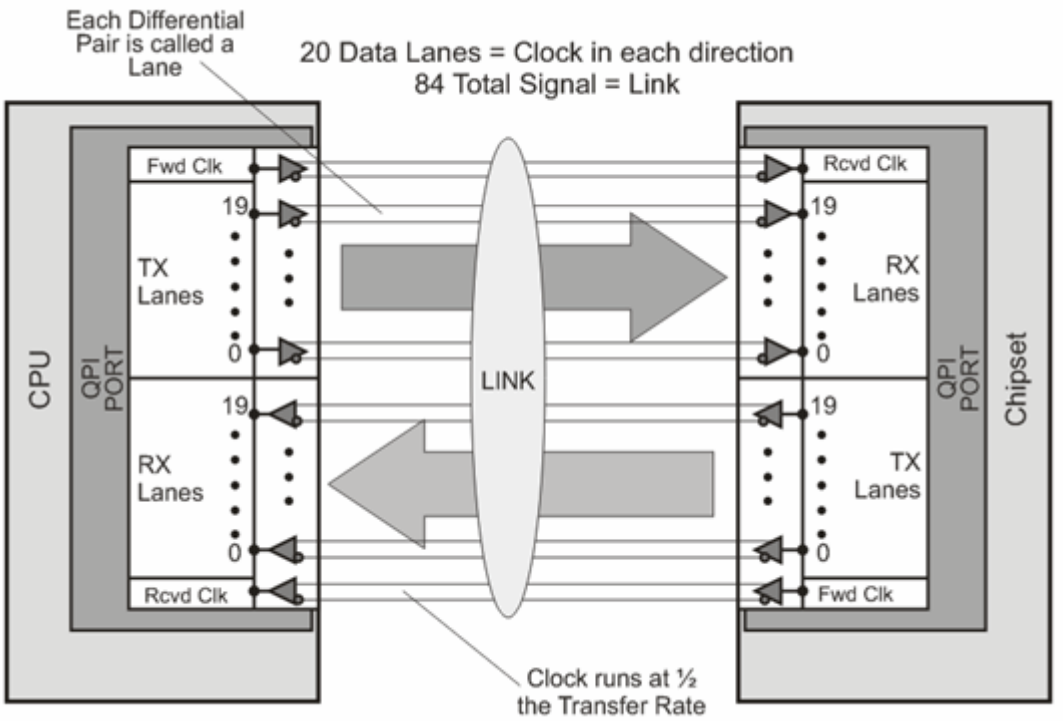


Рис. 51 – Принципы организации шины *Quick Path Interconnect* (каждую отдельную дифференциальную пару называют линией)

На физическом уровне шина образована двумя парами проводников: одна пара служит для передачи данных, а вторая – для их приема. Две такие пары позволяют организовать двунаправленную линию передачи данных, если полосы пропускания недостаточно, то для обеспечения большей пропускной способности используется не одна, а несколько таких двунаправленных линий связи. Физический уровень содержит все необходимые схемы для выполнения интерфейсных операций обмена данными, включая драйвер и входные/входные буферы, параллельно-последовательное и последовательно-параллельное преобразование, схемы ФАПЧ и согласования импеданса. Кроме того, он выполняет также логические функции, связанные с инициализацией и поддержкой интерфейса: такие как конфигурирование канала связи, управление шириной информационной магистрали в операции обмена. Физический интерфейс шины основан на низковольтных драйверах дифференциальных сигналов (рис.52).

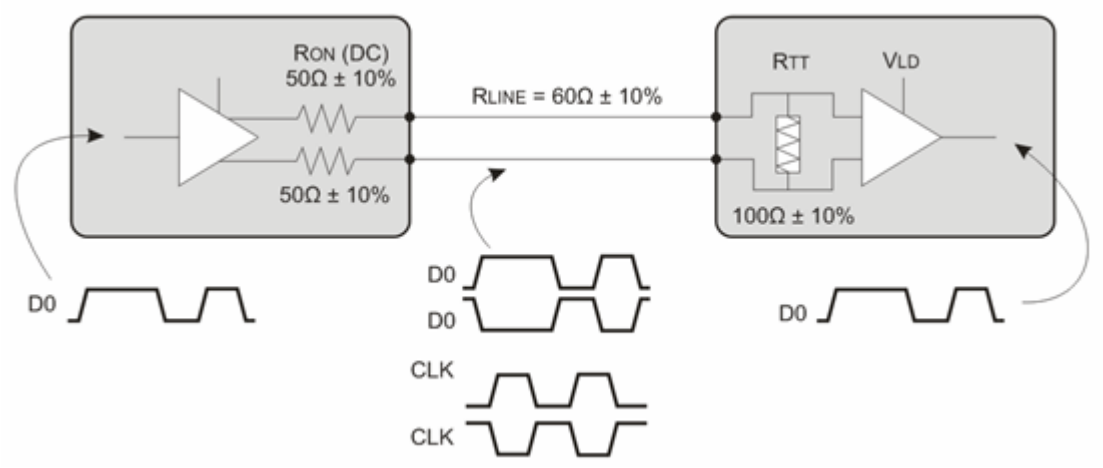


Рис. 52 – Принципы физической реализации линий связи шины

Аналоговая часть интерфейса управляет передачей цифровых данных: при передаче вырабатывает аналоговые уровни сигналов с надлежащим формированием временных

интервалов относительно сигнала синхронизации, а при приеме сигналов на другом конце линии преобразовывает их обратно в цифровые данные. Эта часть интерфейса непосредственно управляет передачей сигналов на проводах шины: один цикл шины (одна операция обмена) выполняется по 20 отдельным парам в дифференциальном виде. В некоторых случаях, связь может осуществляться в половине или четверти ширины шины, например, чтобы уменьшить расход энергии или из-за отказов на линии.

Единицу информации, переданной в каждой единице времени физическим уровнем называют *phit* (можно говорить кадр). Типичные скорости передачи сигналов равны 6.4 ГТ/сек для систем с короткими линиями связи между компонентами, и 4.8 ГТ/сек для более длинных линий, используемых в больших мультипроцессорных системах. Управлением потоком данных занимается специальный «агент», который распределяет поток данных перед тем, как отправить его по различным физическим линиям, а при приеме аналогичный агент собирает разные потоки данных в один.

Для обмена информацией между компонентами системы используются пакеты. Пакеты формируются для обеспечения надежности передачи информации от передающего к принимающему компоненту системы, а также содержат вспомогательные данные, необходимые для обработки пакета на соответствующем уровне интерфейса.

Физический уровень принимает с линий связи кадр проверяет его корректность и выделяет из него пакет. Физическим уровнем биты *phits* и биты контроля циклического избыточного кода не контролируются. Физический уровень объединяет *phits* в пакеты, и передает их на вышестоящий уровень (слой) связи. Каждый пакет, состоит из 80 бит (рис.53).

Рис. 53 – Обобщенная структура пакета и его состав для разных уровней интерфейса

7.2 Интерфейсы. *USB*

Последовательные интерфейсы на протяжении фактически всей истории развития ЭВМ постепенно завоевывали внимание разработчиков по вполне объективным показателям:

- количество физических линий минимально (себестоимость меньше);
- помехоустойчивость выше (значит при всех прочих параметрах надежность передачи выше, длина линии связи больше, пропускная способность выше).

Действительно, история развития последовательного связного интерфейса ПЭВМ (табл.15) позволяет в этом убедиться.

Таблица 15. Хронология развития последовательного порта

год	описание
1962	Разработан телекоммуникационный стандарт *RS*-232, являющийся универсальным стандартом обмена последовательными данными в синхронном или асинхронном режимах (кабель содержал 26 контактов). Скорость обмена достигала 130 Кбод, дальность связи до 7.6 м
1983	На основе *RS*-232 *IBM* и *Intel* разработали универсальный порт для ПК (известен как *COM* порт), модификация получила название *RS*-232*C*: асинхронный, дуплексный, 10-и проводной интерфейс типа точка-точка, обеспечивающий скорость связи до 115200 бод (в каждую сторону) на расстояние до 7.6 м
1983	Разработан стандарт *RS*-485, позволяющий работать через *COM* порт на большие расстояния со многими абонентами: это 4-х проводной, полудуплексный, асинхронный интерфейс со скоростью передачи до 10 Мбод и дальностью связи до 1.2 км
1991	Увеличена базовая частота *COM* порта до 24 МГц, что повысило скорость обмена до 1.5 Мбод (с учетом дуплекса до 3 Мбод)
1996	Разработана спецификация универсального порта *USB* 1.0, который представлял собой 5-и проводной, асинхронный, полудуплексный интерфейс со скоростью обмена до 12 Мбит/с (длина пакета 4-1030 байт) и дальностью связи до 5 м.
2000	Разработана спецификация *USB* 2.0, обеспечивающая скорости до 480 Мбит/с (имеется обратная совместимость: устройства стандарта 1.х могут работать в порту версии 2.0): • низкая скорость (*LS*) - 1.5 Мбит/с; • полная скорость (*FS*) - 12 Мбит/с; • высокая скорость (*HS*) - 480 Мбит/с.
2008	Разработана спецификация *USB* 3.0, пропускная способность 5 Гбит/с (теоретически это соответствует 640 Мбайт/с без учета трафика служебной информации).

USB создавался на основе многолетнего опыта использования *COM* порта ПК и очень удачной связки физических уровней *RS*-232 и *RS*-485 – рис.54.

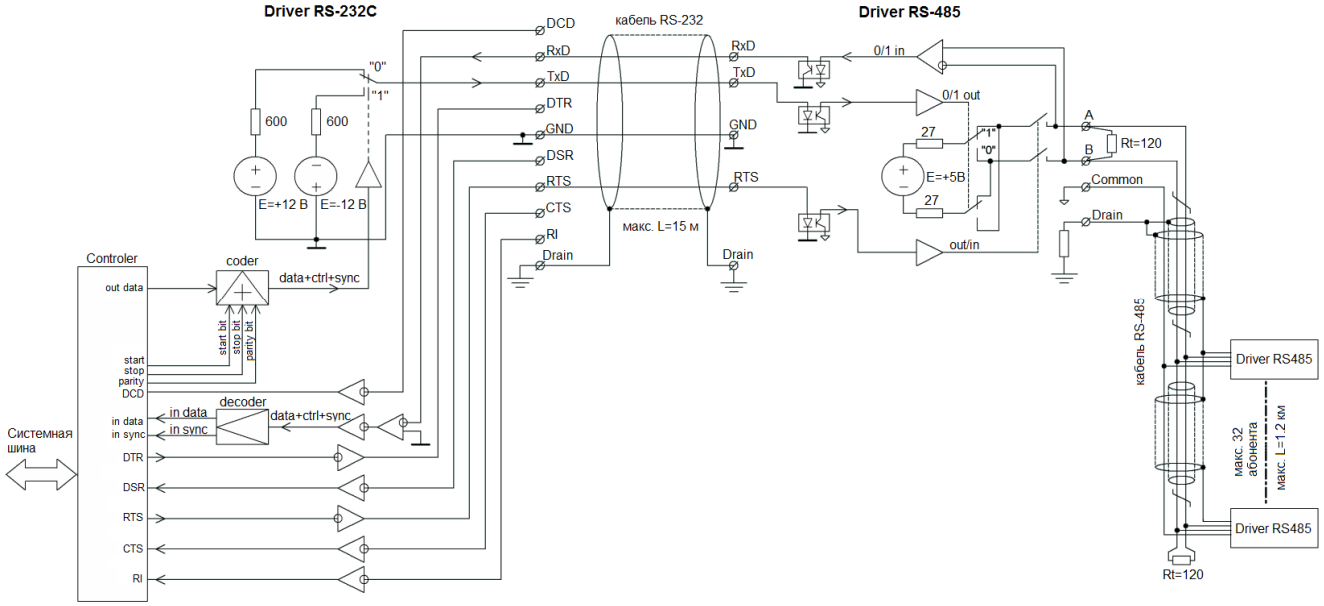


Рис. 54 – Связка *RS*-232*C* и *RS*-485

Физический уровень *USB* напоминает таковой *RS*-485 – применяется аналогичный полудуплексный балансный сигнал, а для уменьшения количества линии связи было решено отказаться от сервисных сигналов, которые управляют обменом данных. Сервисные сигналы *RS*-232 были заменены состояниями, которые передаются по балансной линии связи – тем самым стало возможным переместить физический уровень *RS*-232 в программную область интерфейса. Это придало интерфейсу дополнительную конфигурационную гибкость и упростило его физическую реализацию, также в интерфейс была добавлена линия питания +5В, что еще увеличило его функциональность. Но, при этом интерфейс лишился гальванической развязки, согласованности линии связи, симметричности. Несогласованность, асимметричность и увеличенная скорость передачи сократили расстояние связи до 5 метров.

Для подключения периферийных устройств используется 4-жильный экранированный кабель: питание +5В, сигнальные провода *D*+ и *D*-, общий провод. Интерфейс *USB* соединяет между собой хост (*host*) и устройства. На примере ПК, хост находится внутри ПЭВМ и управляет работой всего интерфейса. Для того, чтобы к одному порту *USB* можно было подключать более одного устройства, применяются хабы (*hub* – устройство, обеспечивающее подключение к интерфейсу других устройств). Корневой хаб (*root hub*) находится внутри ПК и подключен непосредственно к хосту. Логически законченные устройства, подключенные к шине и выполняющие какую-либо специфическую функцию, в интерфейсе *USB* называются "функциями". Топология интерфейса *USB* представляет собой набор из 7 уровней: на первом уровне находится хост и корневой хаб, а на последнем – только функции – рис.55. Устройство, в состав которого входит хаб и одна или несколько функций, называется составным. Порт хаба или функции, подключаемый к хабу более высокого уровня, называется восходящим портом (*upstream port*), а порт хаба, подключаемый к хабу более низкого уровня или к функции называется нисходящим портом (*downstream port*).

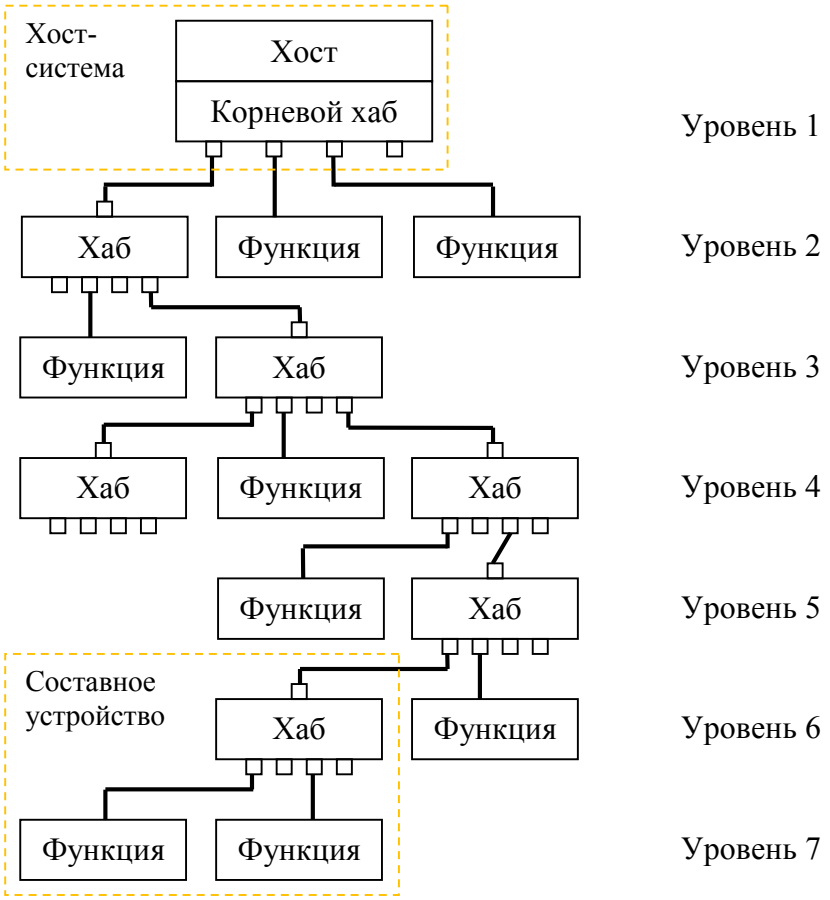


Рис. 55 – Топология *USB* интерфейса

Физический уровень *USB* состоит из двух драйверов и физического кабеля между ними. Драйверы физического уровня несимметричны (имеют разную структуру) и обеспечивают полудуплексную связь. Поэтому существует два типа драйвера:

- *Downstream* (вниз передающий) – этот драйвер всегда ведущий, т.е. только он определяет кто, когда и сколько будет передавать данных в линию связи. Драйверы этого типа всегда генерируют информационный сигнал в направлении от хоста. Эти драйверы установлены на хосте или внизпередающем порту хаба.
- *Upstream* (вверхпередающий) – это драйвер всегда ведомый. Он всегда генерирует информационный сигнал в направлении хоста. Время и порядок его работы определяет ведущий драйвер (*Downstream*). Эти драйвера устанавливаются в устройствах и верхпередающих портах хаба.

Так как драйверы интерфейса несимметричны, то связь на физическом уровне *USB* возможна только между драйверами разного типа, то есть только между *Downstream* и *Upstream*. Соответственно несимметричен и кабель для подключения, со стороны *Downstream* он имеет разъём серии *A*, а со стороны *Upstream* разъём серии *B*. На рис.56 показан драйвер *USB* порта.

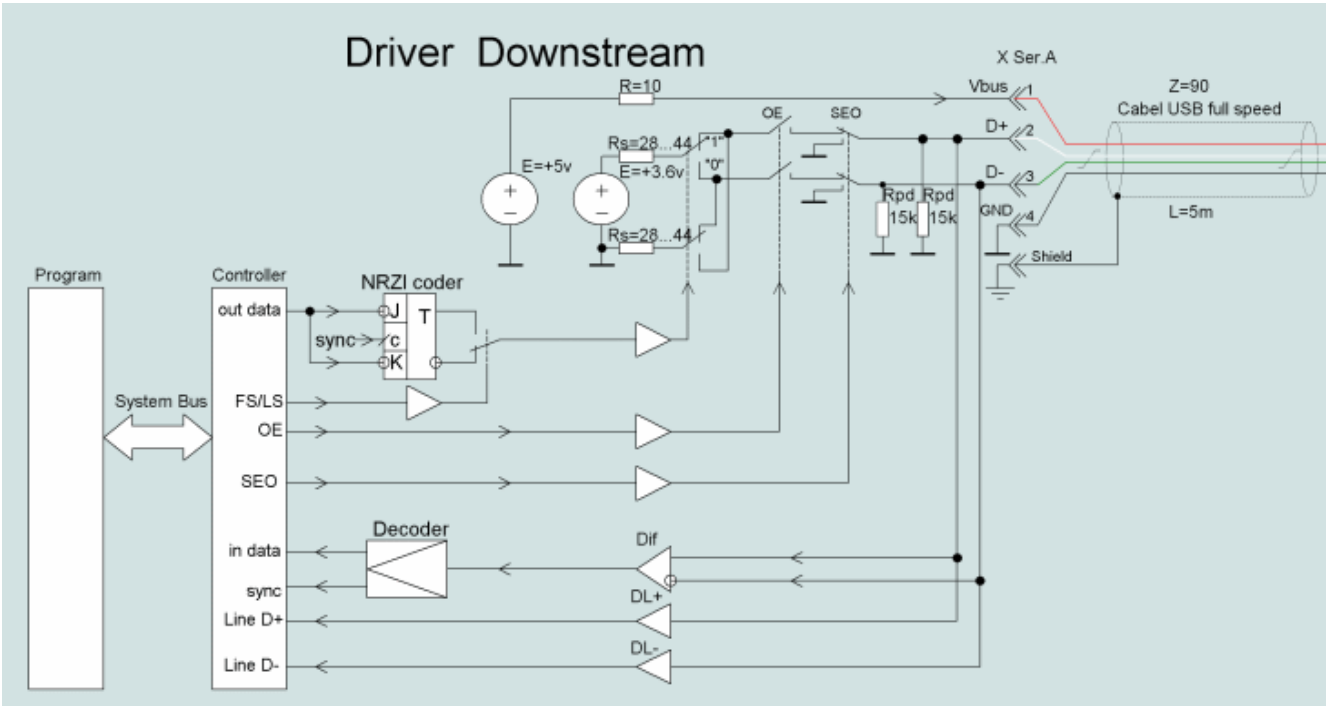


Рис. 56 – Драйвер нисходящего потока

Для низкоскоростных и полноскоростных устройств дифференциальная лог. "1" передается путем подтяжки линии *D*+ к напряжению более 2.8 В, а линии *D*- к напряжению менее 0.3 В. При этом линии *D*+ и *D*- терминированы на стороне нисходящего потока резисторами 15 кОм. Скорость, используемая устройством, подключенным к конкретному порту, определяется хабом по уровням сигналов *D*+ и *D*-, смещаемых нагрузочными резисторами приемопередатчиков: устройства с низкой скоростью (*LS* устройства) подтягивают к высокому уровню линию *D*-, с полной (*FS* устройства) – линию *D*+. Подключение *HS* устройств определяется на этапе конфигурирования.

При передаче данных по шине в виде *LVDS* сигналов все данные кодируются с помощью метода *NRZI* (*Non Return to Zero Invert* – метод возврата к нулю с инвертированием единиц) *with bit stuffing* (с вставляемыми битами синхронизации). Вместо кодирования логических уровней как уровней напряжения, интерфейс *USB* определяет лог. "0" как изменение напряжения, а лог. "1" как неизменение напряжения (рис.57):

- если текущий бит имеет значение "0", то текущий потенциал представляет собой инверсию потенциала предыдущего бита, независимо от его значения;
- если же текущий бит имеет значение "1", то текущий потенциал повторяет предыдущий.

Аппаратное кодирование по методу *NRZI* легко реализуется с помощью *JK*-триггера (см. рис.56). Для того чтобы не потерять синхронизацию при монотонном единичном сигнале производится вставка нуля после каждых 6 единичных битовых интервалов независимо от информационного сигнала. На приемной стороне при наличии 6 единичных битовых интервалов подряд следующий ноль удаляется аппаратно. Этот процесс называется *bit stuffing*.

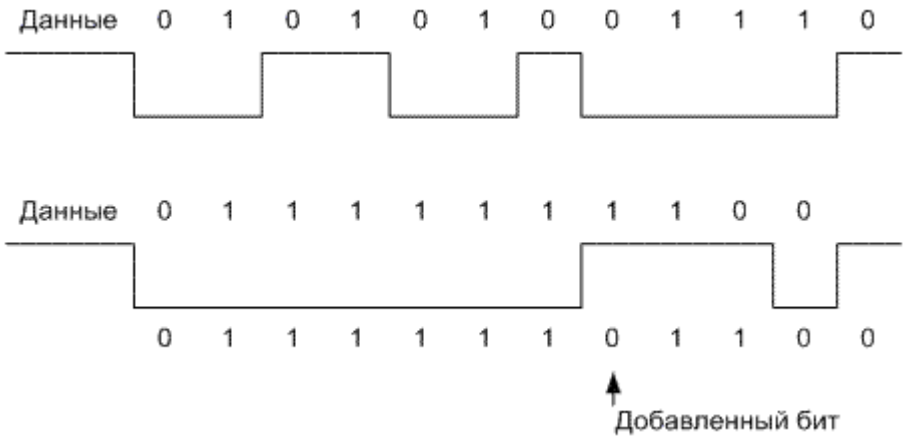


Рис. 57 – Пример *NRZI* кодирования

Интерфейс *USB* обеспечивает одновременный обмен данными между хостом и множеством периферийных устройств. Распределение пропускной способности шины между устройствами планируется хостом и реализуется им с помощью посылки маркеров. Шина позволяет подключать, конфигурировать, использовать и отключать устройства во время работы хоста и самих устройств. Устройство *USB* должно обеспечивать полную поддержку протокола *USB*: выполнение стандартных операций (конфигурирование и сброс), предоставление информации, описывающей устройство. Устройство (хаб или функция) представляется набором конечных точек (от 1 до 16 шт.), каждая точка является отдельным каналом связи между хостом и устройством. Через канал по умолчанию хост получает информацию об устройстве (дескриптор устройства), присваивает устройству адрес (от 0 до 127) и записывает байт конфигурации устройства (после завершения "опроса" всех конечных точек этого устройства). В каждом устройстве есть нулевая конечная точка, которая по умолчанию формирует канал с типом передачи *Control* (типы передачи: *Control, Bulk, Interrupt, Isochronus*). Определенный набор сконфигурированных конечных точек используемых совместно и называется интерфейсом. Устройство может иметь несколько интерфейсов, которые описываются в конфигурации. Таким образом, у устройства может быть несколько конфигураций, каждая из которых может иметь один или несколько интерфейсов, а каждый интерфейс может иметь одну или несколько конечных точек. Для описания атрибутов устройства применяются дескрипторы, которые сообщаются устройством по запросу хоста *Device Request*. Дескрипторы и запросы хоста объединяются в протоколы, подклассы и классы, образуя иерархическую базу (структуру) данных:
Класс-Подкласс-Протокол-Конфигурация-Интерфейс-Конечная точка.

Итак, конечная точка (*Endpoint*) это неотъемлемая часть *USB* устройства, которая имеет уникальный идентификатор и является получателем или отправителем информации, передаваемой по шине (другими словами это буфер, сохраняющий несколько байт). Данные, хранящиеся в конечной точке, могут быть либо принятыми данными, либо данными, ожидающими передачу. Хост также имеет буфер для приема и передачи данных, но хост не имеет конечных точек. Конечная точка имеет следующие основные параметры:

- частота доступа к шине;
- допустимая величина задержки обслуживания;
- требуемая ширина полосы пропускания канала;
- номер конечной точки;
- способ обработки ошибок;
- максимальный размер пакета, который конечная точка может принимать или отправлять;

- используемый конечной точкой тип посылок;
- направление передачи данных.

Все передачи данных по интерфейсу инициируются только хостом. Данные передаются в виде пакетов. Начало пакета определяется по сигналу *SOP* (*Start of Packet*) – это переход линии из состояния *Idle* в состояние "*K*" (рис.56). Далее следует байт синхронизации *SYNC* (80h), который поле *NRZI* кодирования имеет вид "*KJKJKJKK*". Последние два символа "*KK*" являются маркером начала блока данных, при этом байты данных передаются начиная с младшего бита и заканчивая старшим (суммарная длина блока данных может быть разной, но кратной байту). Каждый пакет содержит поле *PID* (8 бит), которое является его идентификатором (причем поле представляет собой 4 бита информации в старшей тетраде и их инверсию в младшей). Завершаются пакеты полем *EOP* – поле «тишины», когда в линию ничего не передается, если хаб обнаружит передачу в этом интервале, то он отключит соответствующий порт и сообщит об этом хосту.

Пакеты подразделяются на несколько видов:

1. **Пакет-признак** (*token packet*), также называемый маркером, описывает тип и направление передачи данных, адрес устройства и порядковый номер конечной точки (адресуемая часть *USB* устройства), пакет-признаки бывают нескольких типов: *IN*, *OUT*, *SOF*, *SETUP*. Этот пакет всегда посылается хостом и является заголовком транзакции, т.е. определяет кому и как будет передаваться информация в следующем пакете.

Sync (8 бит) 80h	*PID* (4 бит)	инв. *PID*	*ADDR* (7 бит) 0-127	*ENDP* (4 бита) 0-15	*CRC* (5 бит)	*EOP* (2 бита)

   - поле *PID*:
     - *OUT* /=0001b/ – передача данных от хоста к конечной точке
     - IN /=1001b/ – передача данных от конечной точки к хосту
     - SETUP /=1101b/ – передача от хоста к конечной точке по каналу управления
   - поле *ADDR* – адрес устройства с которым будет работать хост в текущей транзакции
   - поле *ENDP* – адрес конечной точки с которой будет работать хост в текущей транзакции
   - поле *PID*=0101b – пакет *SOF* (*Start of Frame Packets*). Этот пакет посылается хостом 1000 раз в секунду и обозначает начало нового кадра, его принимают все устройства одновременно (можно использовать для синхронизации устройства и хоста). Содержит поле *Frame number* – счетчик кадров (фактически это счетчик миллисекунд).

Sync (8 бит) 80h	*PID*= 0101b	1010b	*Frame number* 0-2047	*CRC* (5 бит)	*EOP* (2 бита)

   - поле *CRC* – контрольная сумма полей *ADDR*, *ENDP* или поля *Frame number* вычисляется побитовой сверткой данных с полиномом 25h (без учёта дополнительных синхробитов (*Stuffed Bit*, которые добавляются и убираются в поток данных аппаратно и служат для синхронизации).

2. **Пакет с данными** (*data packet*) содержит данные, которыми обмениваются хост и конечная точка устройства, в транзакции всегда следует за маркерным пакетом;

Sync (8 бит) 80h	*PID* (4 бит)	инв. *PID*	*Data* *LS*: 0-8 байт *FS*: 0-1023 байта	*CRC* (16 бит)	*EOP* (2 бита)

   - Поле *PID*:
     - Data0 /=0011b/ – пакет данных с четным номером
     - Data1 /=1011b/ – пакет данных с нечетным номером

     (*) чередование *PID* помогает контролировать правильность передачи.
   - Поле *CRC* вычисляется побитовой сверткой данных *Data* и полинома 18005h (без учёта дополнительных синхробитов *Stuffed Bit*)

3. **Пакет согласования** (*handshake packet*) предназначен для сообщения о результатах пересылки данных, пакеты согласования бывают нескольких типов: *ACK*, *NAK*, *STALL*. Этот пакет завершает транзакцию между хостом и конечной точкой.

Sync (8 бит) 80h	*PID* (4 бит)	инв. *PID*	*EOP* (2 бита)

- Поле *PID*:
  - ACK /=0010b/ – выставляется хостом (*IN* транзакция) или функцией (*OUT* транзакция) и подтверждает, что обмен данными между ними прошел без ошибок
  - NAK /=1010b/ – выставляется только функцией, подтверждает что данные *OUT* транзакции приняты от хоста с ошибкой
  - STALL /=1110b/ – выставляется только функцией, говорит о не готовности функции к обмену в *IN*/*OUT* транзакциях.

4. **Специальные**, например: *Preamble Packet* – преамбула низкоскоростного пакета. Этот пакет предписывает хабу, работающему на полной скорости, передать следующий за преамбулой пакет на низкой скорости. После получения сигнала *EOP* в конце низкоскоростного пакета хаб должен вернуться на полную скорость.

Sync (8 бит) 80h	*PID*= 1100	 0011

Таким образом, каждая транзакция состоит из трех фаз: фаза передачи пакета-признака (маркера), фаза передачи данных и фаза согласования (подтверждения).

Транзакция – это однократный обмен данными между хостом и конечной точкой устройства. В интерфейсе *USB* различают следующие типы транзакций:

1. **Управляющая транзакция** (*control*) используется для конфигурации устройства, а также для других специфических для конкретного устройства целей (хост посылает, конечная точка принимает). Транзакция содержит *Setup* пакет, необязательный пакет с данным и пакет согласования – рис.58. Если пакет с данными получен конечной точкой успешно, то она отсылает хосту *ACK* пакет. В противном случае транзакция завершается. В стадии передачи данных управляющие пересылки содержат одну или несколько *IN* или *OUT* транзакций, принцип передачи которых такой же, как и в потоковых пересылках (все транзакции в стадии передачи данных должны производиться в одном направлении).

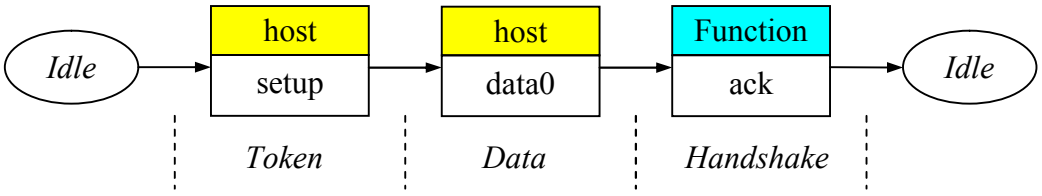


Рис. 58 – Транзакция *Setup*

2. **Потоковая транзакция** (*bulk*) используется для передачи относительно большого объема информации. Транзакция характеризуются гарантированной безошибочной передачей данных между хостом и конечной точкой посредством обнаружения ошибок при передаче и повторного запроса информации – рис.59. Когда хост становится готовым принимать данные от функции, он в фазе передачи пакета-признака посылает функции *IN* пакет. В ответ на это функция в фазе передачи данных передает хосту пакет с данными или, если она не может сделать этого, передает *NAK* или *STALL* пакет (*NAK* пакет сообщает о временной неготовности функции передавать данные, а *STALL* пакет сообщает о необходимости вмешательства хоста). Если хост успешно получил данные, то он в фазе согласования посылает функции *ACK* пакет. В противном случае транзакция завершается. Когда хост становится готовым передавать данные, он посылает функции *OUT* пакет, сопровождаемый пакетом с данными. Если функция успешно получила данные, она отсылает хосту *ACK* пакет, в противном случае отсылается *NAK* или *STALL* пакет.

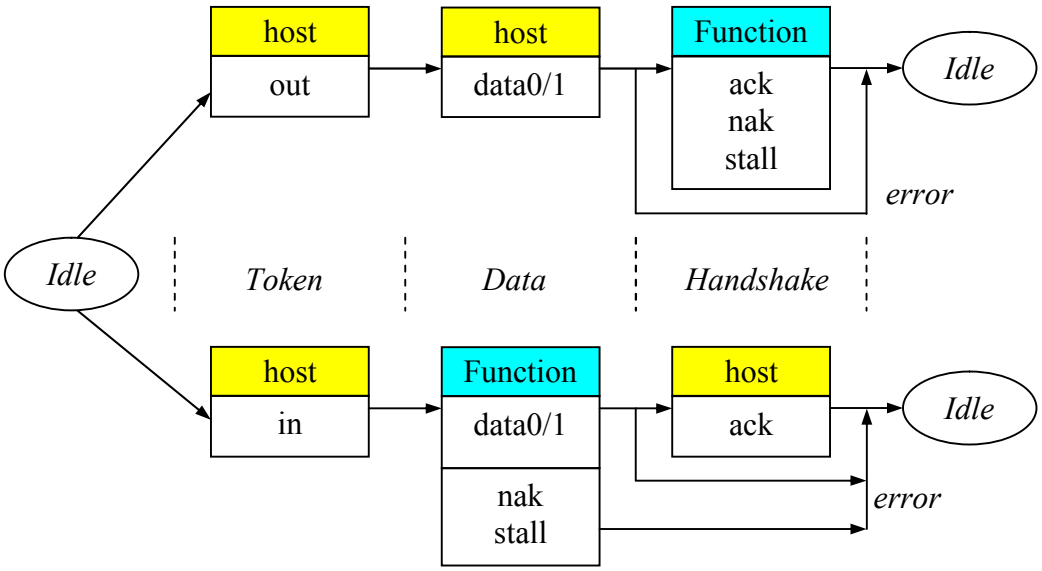


Рис. 59 – Транзакция *Bulk*

3. **Транзакция с прерыванием** (*interrupt*) используется для передачи относительно небольшого объема информации, для которого важна своевременная его пересылка (имеет ограниченную длительность и повышенный приоритет относительно других типов пересылок). Транзакция может содержать *IN* или *OUT* пересылки. При получении *IN* пакета функция может вернуть пакет с данными, *NAK* пакет (если у функции нет информации, для которой требуется прерывание) или *STALL* пакет (если работа конечной точки с прерыванием приостановлена). При необходимости прерывания функция возвращает необходимую информацию в фазе передачи данных. Если хост успешно получил данные, то он посылает *ACK* пакет. В противном случае согласующий пакет хостом не посылается. В целом алгоритм пересылок аналогичен случаю потоковой транзакции.

4. **Изохронная транзакция** (*isochronous*) также называется потоковой пересылкой реального времени (микрофон, видео и т.п.). Этот тип пересылок гарантирует время доставки данных, но не гарантирует их правильность. Транзакция содержит фазу передачи маркера и фазу передачи данных, но не имеют фазы согласования – рис.60. Хост отсылает *IN* или *OUT* признак, после чего в фазе передачи данных конечной точке (для *IN* пакета) или хост (для *OUT* пакета) пересылает данные.

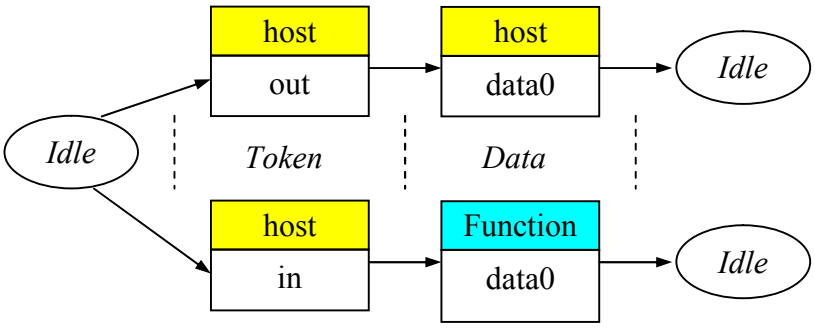


Рис. 60 – Транзакция *isochronous*

В заключении отметим, что

- время ожидания в транзакциях: устройство должно ожидать квитирования не менее 16 битовых интервалов и приостанавливаться через 18 битовых интервалов; хост может посылать следующий маркер (если устройство не ответило) не ранее чем через 18 битовых интервалов;

- ограничения для *LS* транзакций: максимальное количество байт данных в транзакции ≤8, поддерживаются только транзакции прерывания и управления, пакет *SOF* не распознается *LS* устройствами.

На рис.61 представлены типичные процессы обмена на шине *USB*.

Рис. 61 – Транзакции передачи пакетов шины *USB* на временной шкале

В связи с тем, что в интерфейсе *USB* реализован сложный протокол обмена информацией, в устройстве сопряжения с интерфейсом *USB* необходим микропроцессорный блок, обеспечивающий поддержку протокола. Поэтому основным вариантом при разработке устройства сопряжения является применение микроконтроллера, который будет обеспечивать поддержку протокола обмена. В настоящее время все основные производители микроконтроллеров выпускают продукцию, имеющую в своем составе блок *USB*.

7.3 Интерфейсы. *SPI*

SPI (*Serial Peripheral Interface*) – последовательный синхронный интерфейс полного дуплекса, предназначенный для обмена данными между микросхемами внутри системы. Интерфейс *SPI* (разработан компанией *Motorola*), наряду с I^2C, относится к самым широко-распространенным. Шина *SPI* организована по принципу "ведущий-подчиненный" (*master-slave*). В качестве ведущего шины обычно выступает процессор (микроконтроллер, *DSP*, ПЛИС и т.п.). Подключенные к ведущему внешние устройства образуют подчиненных шины – обычно это различного рода микросхемы: запоминающие устройства (*EEPROM*, *Flash*-память, *SRAM*), часы реального времени (*RTC*), АЦП/ЦАП и др. В отличие от стандартного последовательного порта (например *RS*-232*C*), *SPI* является синхронным интерфейсом, в котором любая передача синхронизирована с общим тактовым сигналом, генерируемым ведущим устройством. Принимающая (ведомая) периферия синхронизирует получение битовой последовательности с тактовым сигналом. К ведущему может присоединяться несколько ведомых, которые в "классическом варианте" разделяются с помощью сигнала «выбор кристалла» (*chip select*) на ведомой микросхеме. Периферия, не выбранная ведущим, не принимает участия в обмене.

В *SPI* используются четыре линии:

- *MOSI* (*Master Out Slave In*) – выход ведущего, вход ведомого. Служит для передачи данных от ведущего устройства ведомому.
- *MISO* (*Master In Slave Out*) – вход ведущего, выход ведомого. Служит для передачи данных от ведомого устройства ведущему.
- *SCLK* (*Serial Clock*) – тактовый сигнал от ведущего к ведомым устройствам.
- *CS* (*Chip Select*) или *SS* (*Slave Select*) – выбор микросхемы, выбор ведомого.

На рис.62 представлены схемы включения устройств к шине *SPI*. Параллельное включение (оно же «классическое») наиболее часто применяется (просто реализуется), его недостатком является необходимость в сигнале выбора микросхемы для каждого устройства (количество линий связи возрастает с ростом количества устройств на шине). Каскадное включение избавлено от такого недостатка, однако такой режим работы должен поддерживаться всеми устройствами, в документации обычно обозначается термином "*daisy-chaining*".

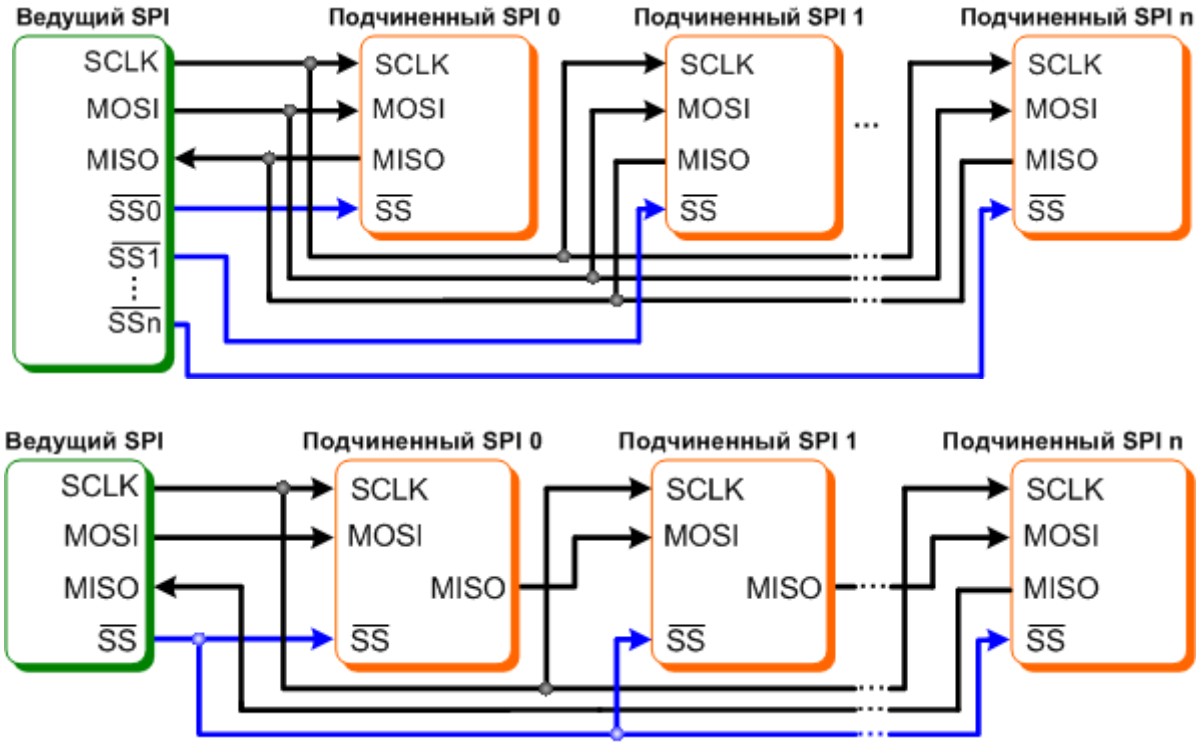


Рис. 62 – Подключение устройств к шине *SPI* (сверху – параллельное или радиальное включение, снизу – каскадное или кольцевое)

Протокол передачи по интерфейсу *SPI* идентичен логике работы сдвигового регистра, которая заключается в выполнении операции сдвига и, соответственно, побитного ввода и вывода данных по определенным фронтам сигнала синхронизации. Установка данных при передаче и выборка при приеме всегда выполняются по противоположным фронтам синхронизации. Это необходимо для гарантирования выборки данных после надежного их установления (длительности переходных процессов меньше длительностей импульса и паузы тактового сигнала). Если к этому учесть, что в качестве первого фронта в цикле передачи может выступать нарастающий или падающий фронт, то всего возможно четыре варианта логики работы интерфейса *SPI*. Эти варианты получили название режимов *SPI* (табл.16) и описываются двумя параметрами:

- *CPOL* – исходный уровень сигнала синхронизации (если *CPOL*=0, то линия синхронизации до начала цикла передачи и после его окончания имеет низкий уровень (т.е. первый фронт нарастающий, а последний – падающий), если *CPOL*=1 – высокий (т.е. первый фронт падающий, а последний – нарастающий);
- *CPHA* – фаза синхронизации, от этого параметра зависит, в какой последовательности выполняется установка и выборка данных (если *CPHA*=0, то по переднему фронту в цикле синхронизации будет выполняться выборка данных, а затем, по заднему фронту – установка данных, если же *CPHA*=1, то установка данных будет выполняться по переднему фронту в цикле синхронизации, а выборка – по заднему).

Таблица 16. Режимы *SPI*

Режим	0	1	2	3
CPOL	0	1	0	1
CPHA	0	0	1	1
Временная диаграмма первого цикла синхронизации