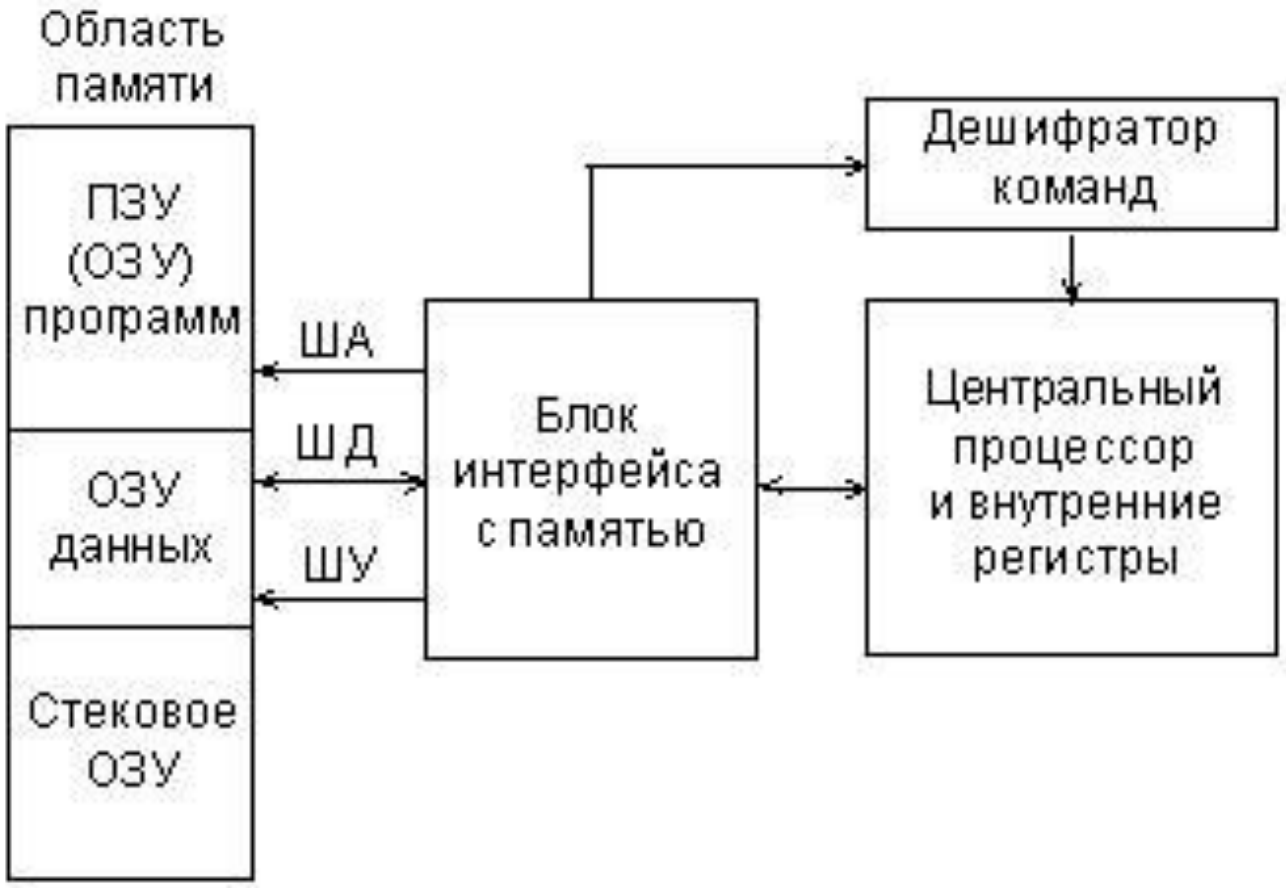
Архитектура МП

Архитектура фон Неймана— широко известный принцип совместного хранения программ и данных в памяти компьютера. Один и тот же подход к рассмотрению данных и инструкций сделал лёгкой задачу изменения самих программ.



«Принципы фон Неймана».

1. **Принцип использования двоичной системы счисления для представления данных и команд**.
2. **Принцип программного управления**.

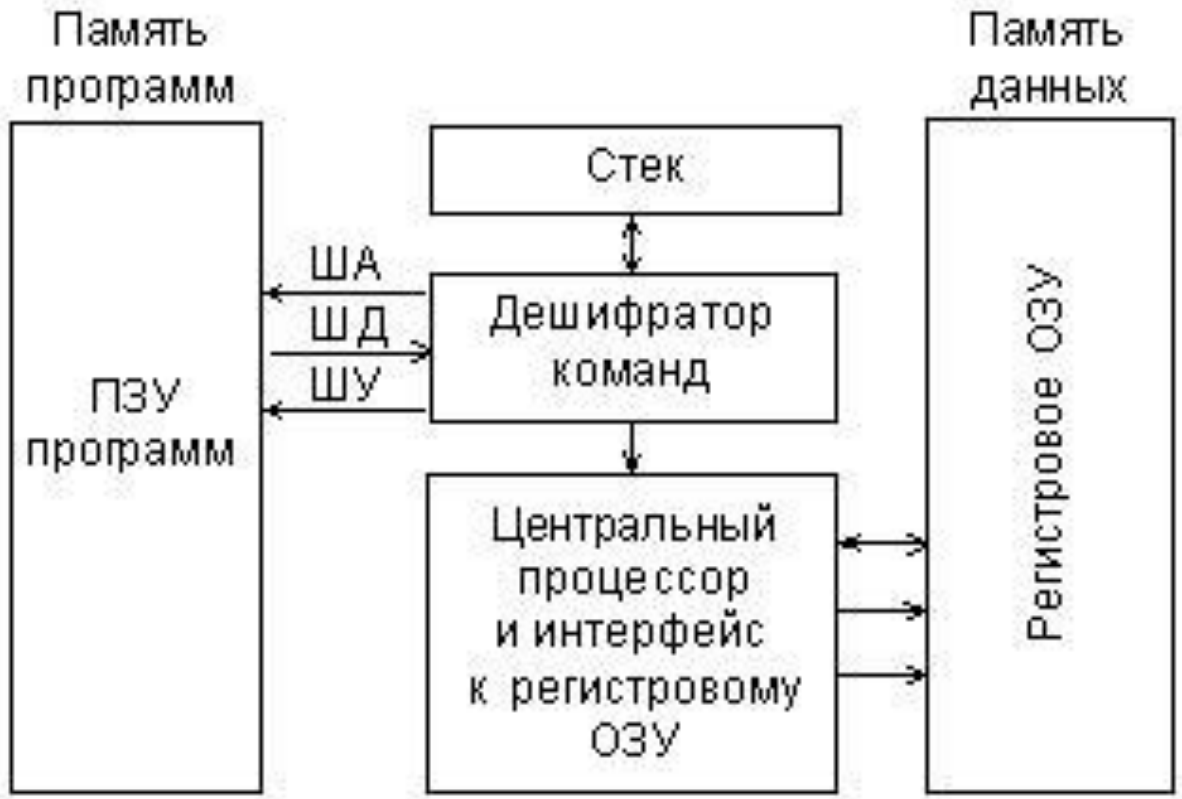
   Программа состоит из набора команд, которые выполняются процессором друг за другом в определенной последовательности.
3. **Принцип однородности памяти**.

   Как программы (команды), так и данные хранятся в одной и той же памяти. Над командами можно выполнять такие же действия, как и над данными.
3. **Принцип адресуемости памяти**.

   Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка.
5. **Принцип последовательного программного управления**
   - Все команды располагаются в памяти и выполняются последовательно, одна после завершения другой.
6. **Принцип условного перехода**.

Гарвардская Архитектура

Гарвардская архитектура — отличительным признаком которой является раздельное хранение и обработка команд и данных . Архитектура была разработана Говардом Эйкеном в конце 1930-годов в Гарвардском университете.



Анализ реальных программ управления показал, что необходимый объем памяти данных МК, используемый для хранения промежуточных результатов, как правило, на порядок меньше требуемого объема памяти программ. В этих условиях использование единого адресного пространства приводило к увеличению формата команд за счет увеличения числа разрядов для адресации операндов. Применение отдельной небольшой по объему памяти данных способствовало сокращению длины команд и ускорению поиска информации в памяти данных.

Кроме того, гарвардская архитектура обеспечивает потенциально более высокую скорость выполнения программы по сравнению с фон-неймановской за счет возможности реализации параллельных операций. Выборка следующей команды может происходить одновременно с выполнением предыдущей, и нет необходимости останавливать процессор на время выборки команды.

CISC -архитектура

Аббревиатура **CISC** означает Complete Instruction Set Computer – компьютер со сложным (полным) набором команд. Несмотря на то, что первый CISC-процессор был разработан компанией IBM (она до сих пор их использует в мейнфреймах типа IBM ES/9000), лидером производства считается компания Intel (i8086, 8286, 8386, и т.д). В CISC мало регистров общего назначения, с которыми можно выполнять арифметические операции, большое количество машинных команд. Это приводит к усложнению декодирования инструкций, что в свою очередь приводит к расходованию аппаратных ресурсов. Слабость CISC архитектуры заключается в том, что 80 процентов вычислений процессора приходилось на 20 процентов команд.

RISC -архитектура

Принцип: более компактные и простые инструкции выполняются быстрее. Простая архитектура позволяет удешевить процессор, поднять тактовую частоту, а также распараллелить исполнение команд между несколькими блоками исполнения (т. н. суперскалярные архитектуры процессоров). Многие ранние RISC-процессоры даже не имели команд умножения и деления. Идея создания RISC процессоров пришла после того, как в 1970-х годах ученые обнаружили, что многие из функциональных особенностей традиционных МП игнорировались программистами или компиляторами. Следующее открытие заключалось в том, что, поскольку некоторые сложные операции использовались редко, они как правило были медленнее, чем те же действия, выполняемые набором простых команд.

Первые RISC-процессоры были разработаны в начале 1980-х годов в Стэнфордском и Калифорнийском университетах. Они выполняли небольшой (50−100) набор команд, тогда как обычные CISC выполняли 100—200.

Характерные особенности RISC-процессоров:

-Фиксированная длина машинных инструкций (например, 32 бита) и простой формат команды.
-Специализированные команды для операций с памятью — чтения или записи. Операции вида «прочитать-изменить-записать» отсутствуют. Любые операции «изменить» выполняются только над содержимым регистров (т. н. load-and-store архитектура).
-Большое количество регистров общего назначения (32 и более).

- Спекулятивное исполнение. При встрече с командой условного перехода процессор исполняет (или по крайней мере читает в кэш инструкций) сразу обе ветви, до тех пор, пока не окончится вычисление управляющего выражения перехода. Позволяет отказаться от простоев конвейера при условных переходах.
- Переименование регистров. Каждый регистр процессора на самом деле представляет собой несколько параллельных регистров, хранящих несколько версий значения.

Как оказалось в начале 1990-х годов, RISC-архитектуры позволяют получить большую производительность, чем CISC, за счет использования суперскалярного и VLIW подхода, а также за счет возможности серьезного повышения тактовой частоты и за счет упрощения кристалла с высвобождением площади под кеш-память. Также RISC-архитектуры позволили сильно снизить энергопотребление процессора за счет уменьшения числа транзисторов (ARM).

MISC процессоры (архитектура)

Minimum Instruction Set Computer — вычисления с минимальным набором команд. Дальнейшее развитие идей команды Чака Мура (1990), который полагает, что принцип простоты, изначальный для RISC-процессоров, слишком быстро отошёл на задний план. В пылу борьбы за максимальное быстродействие, RISC догнал и перегнал многие CISC процессоры по сложности.

Элементная база состоит из двух частей, которые либо выполнены в отдельных корпусах, либо объединены. Основная часть – RISC CPU, обрабатывающий в себе от 10 базовых команд, расширяемый подключением второй части – ПЗУ микропрограммного управления. Система приобретает свойства CISC. Основные команды работают на RISC CPU, а команды расширения преобразуются в адрес микропрограммы. RISC CPU выполняет все команды за один такт, а вторая часть эквивалентна CPU со сложным набором команд. Наличие ПЗУ устраняет недостаток RISC, выраженный в том, что при компиляции с языка высокого уровня микрокод генерируется из библиотеки стандартных функций, занимающей много места в ОЗУ. Поскольку микропрограмма уже дешифрована и открыта для программиста, то времени выборки из ОЗУ на дешифрацию не требуется. Причиной, по которой данная архитектура **не стала популярной** в компьютерных технологиях – **сложность написания программ** под различные процессоры. Ведь все нюансы по подбору методов вычисления и оптимизаций возлагались на плечи программистов.

Классификация Флинна

	Одиночный поток команд	Множество потоков команд
	(Single Instruction)	(Multiple Instruction)
Одиночный поток данных	SISD	MISD
(Single Data)		
Множество потоков данных	SIMD	MIMD
(Multiple Data)		

Архитектура i86

x86 (англ. *Intel 80x86*) — архитектура процессора и одноимённый набор команд, впервые реализованные в процессорах компании Intel.
Название образовано от двух цифр, которыми заканчивались названия процессоров Intel ранних моделей — 8086, 80186, 80286 (i286), 80386 (i386), 80486 (i486). За время своего существования набор команд постоянно расширялся, сохраняя совместимость с предыдущими поколениями.
Первый 16-битный микропроцессор компании Intel. Разрабатывался с весны 1976 года и выпущен 8 июня 1978 года Система команд процессора Intel 8086 состоит из 98 команд (и более 3800 их вариаций): 19 команд передачи данных, 38 команд их обработки, 24 команды перехода и 17 команд управления процессором. Возможны 7 режимов адресации.
Микропроцессор не содержит команд для работы с числами с плавающей запятой. Данная возможность реализована отдельной микросхемой, называемой математический сопроцессор, (например, модель Intel 8087), который устанавливается на материнской плате.
x86 — это CISC-архитектура. Доступ к памяти происходит по «словам». «Слова» размещаются по принципу little-endian, известному также как Intel-формат. Современные процессоры включают в себя декодеры команд x86 для преобразования их в упрощённый внутренний формат с последующим их выполнением.
Помимо Intel архитектура также была реализована в процессорах других производителей: AMD, VIA, Transmeta, IDT и др. В настоящее время для 32-разрядной версии архитектуры существует ещё одно название — IA-32 (Intel Architecture — 32).

Сегментная организация памяти

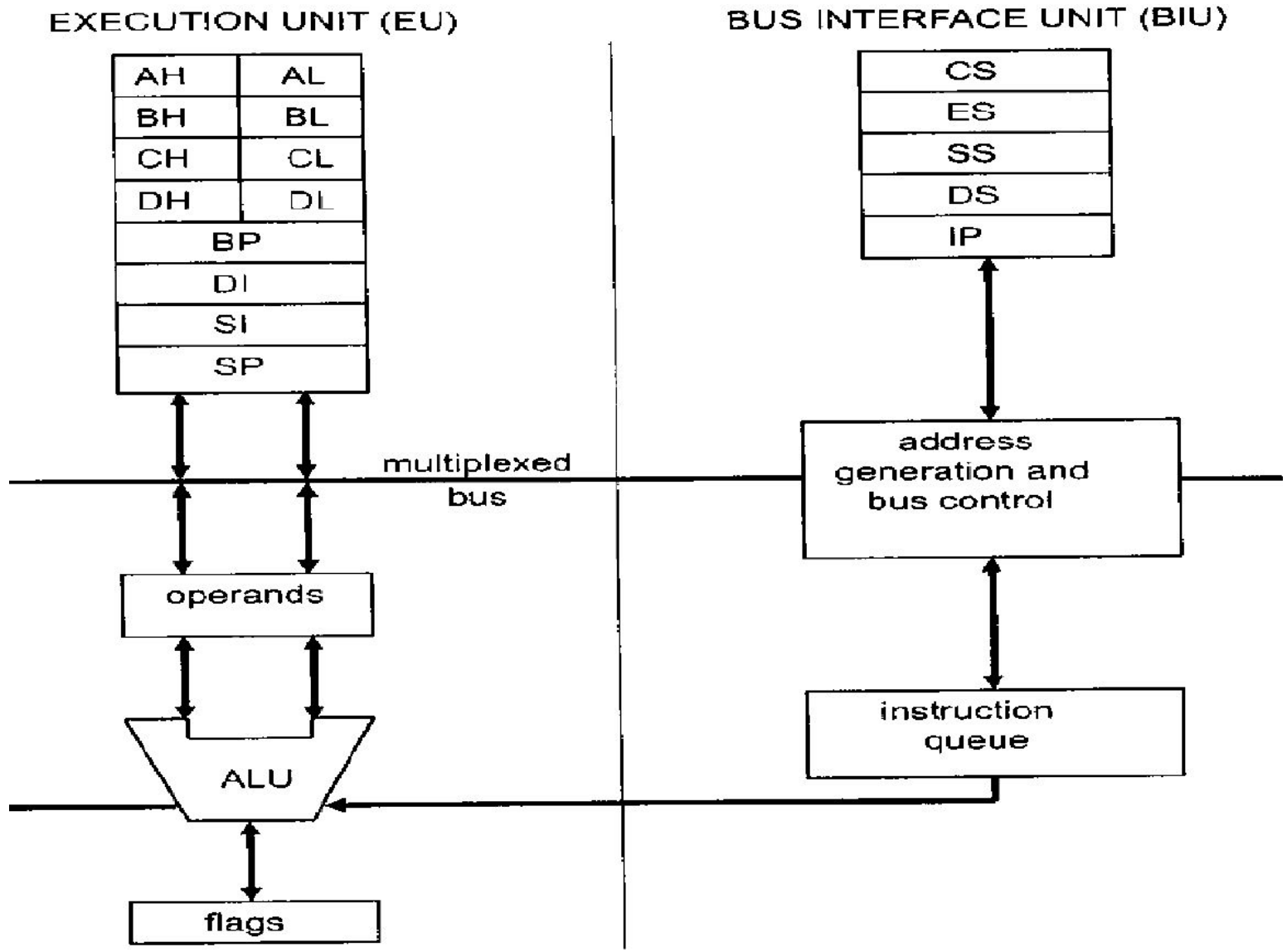
Реальный режим (real mode)

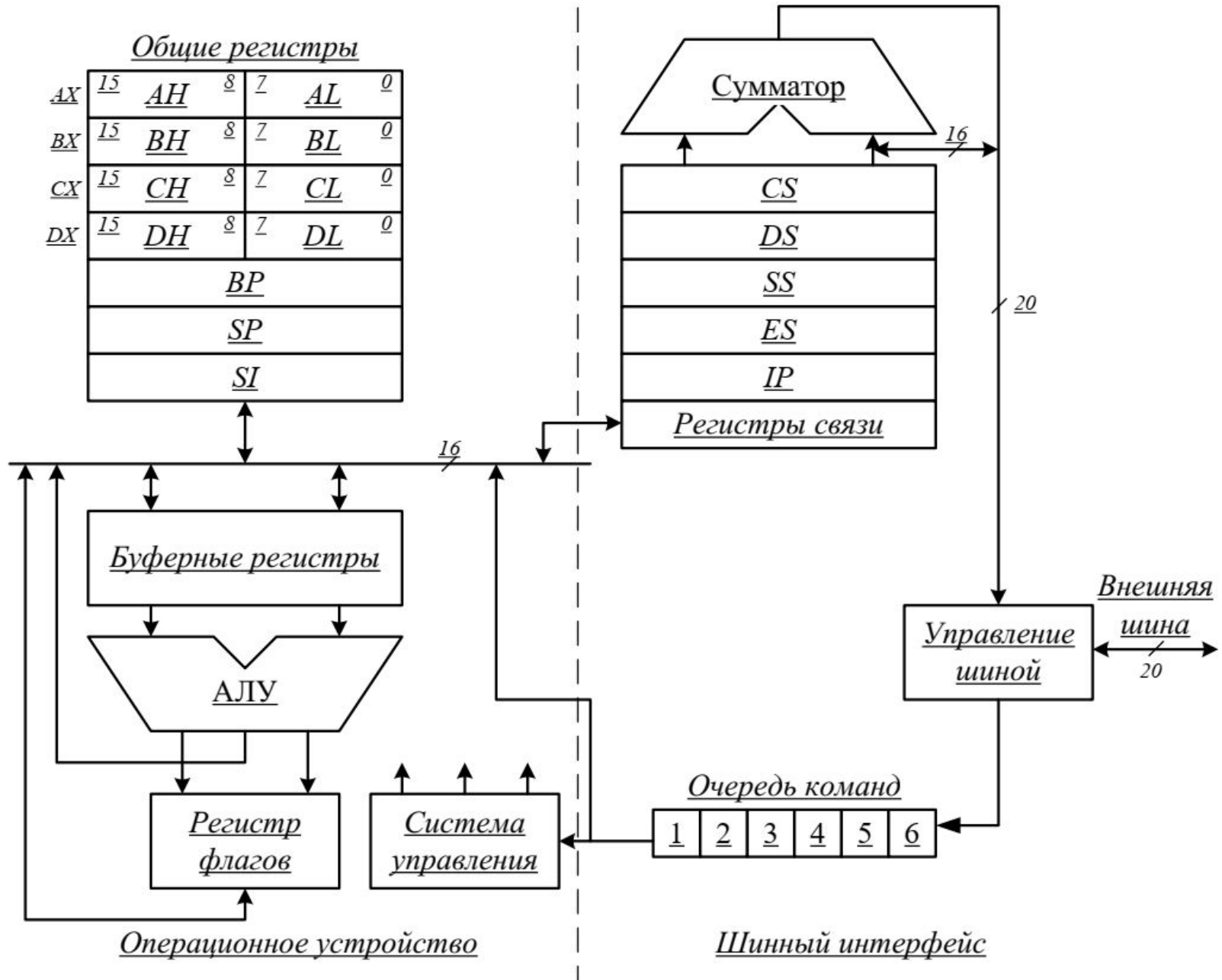
Реальный режим — классический режим адресации, использованный в первых моделях семейства. Адрес ячейки памяти (для работы с данными или для загрузки исполняемой команды процессора) формируется из сегмента (содержимого сегментного регистра) и оффсета-смещения (константа, регистр, сумма регистра с константой или сумма двух регистров с константой); это записывается в виде логического адреса SSSS:OOOO (Segment:Offset), где S и O — шестнадцатеричные цифры. Физический адрес вычисляется по формуле «Segment*16 + Offset».

Защищённый режим (protected mode)

Более совершенный режим, впервые появившийся в процессоре 80286 и в дальнейшем многократно улучшавшийся. Имеет большое количество подрежимов, по которым можно проследить эволюцию семейства ЦП. В этом режиме поддерживается защита памяти, контексты задач и средства для организации виртуальной памяти. Аналогично реальному режиму, тут также используется сегментированная модель памяти, однако уже организованная по другому принципу: деление на параграфы отсутствует, а расположение сегментов описывается специальными структурами (таблицами дескрипторов), расположенными в оперативной памяти.

Упрощенная модель МП




Общие регистры
AX
15
AH
8
7
AL
0
BX
BH
BL
CX
CH
CL
DX
DH
DL
BP
SP
SI
16
Буферные регистры
АЛУ
Регистр флагов
Система управления
Операционное устройство
Сумматор
16
CS
DS
SS
ES
IP
Регистры связи
20
Управление шиной
Внешняя шина
20
Очередь команд
1
2
3
4
5
6
Шинный интерфейс

Сравнение методов программирования

	Вся программа на Си	Вся программа на ассемблере	Программа на Си с фрагментом на ассемблере
Время программирования(в условных единицах)	100	500	180
Время выполнения программы (в условных единицах)	100	20	36

Цикл выполнения команды

Первый этап – *выборка* исполняемой команды из *ОП*.

Второй этап – *выборка* первого операнда

Третий этап – *выборка* второго операнда

Четвертый этап – выполнение *операции*

Пятый этап – обращение к *ОП* и *запись* результата *операции*

Шестой этап – формирование в счетчике команд адреса ячейки *ОП*, где находится следующая *команда* программы

Регистры МП

Table 1-2: Registers of the 8086/286 by Category

Category	Bits	Register Names
General	16	AX, BX, CX, DX
	8	AH, AL, BH, BL, CH, CL, DH, DL
Pointer	16	SP (stack pointer), BP (base pointer)
Index	16	SI (source index), DI (destination index)
Segment	16	CS (code segment), DS (data segment), SS (stack segment), ES (extra segment)
Instruction	16	IP (instruction pointer)
Flag	16	FR (flag register)

Магистрально-модульный принцип построения МПС

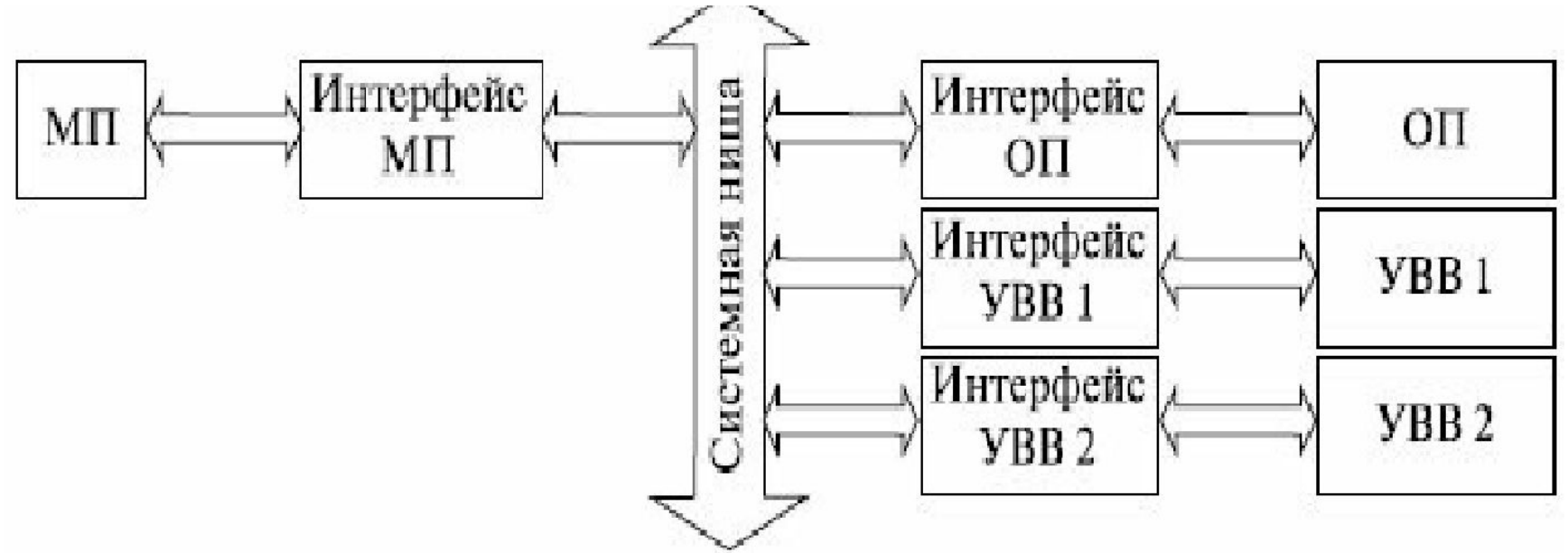
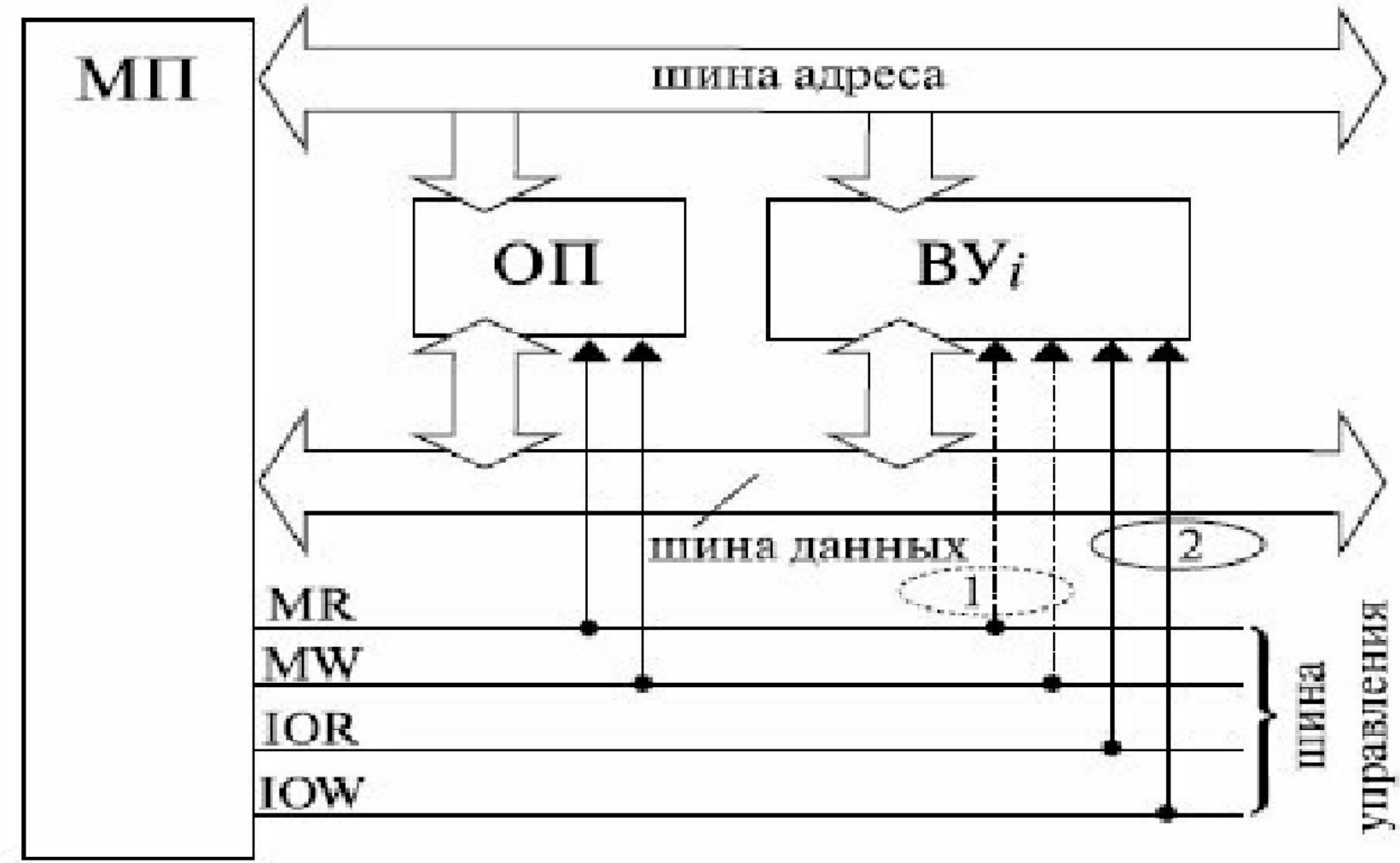


Рис. 8.1. Магистрально-модульный принцип построения микропроцессорной системы

Взаимодействие МП и модулей



2 команды взаимодействия МП и модулей

- команда ввода `IN AX, DX` записывает в регистр `AX` число из внешнего устройства, адрес которого находится в регистре `DX` ; при этом вырабатывается сигнал IOR (`INput/OUTput Read`).);
- команда вывода `OUT DX, AX` выводит информацию из регистра `AX`

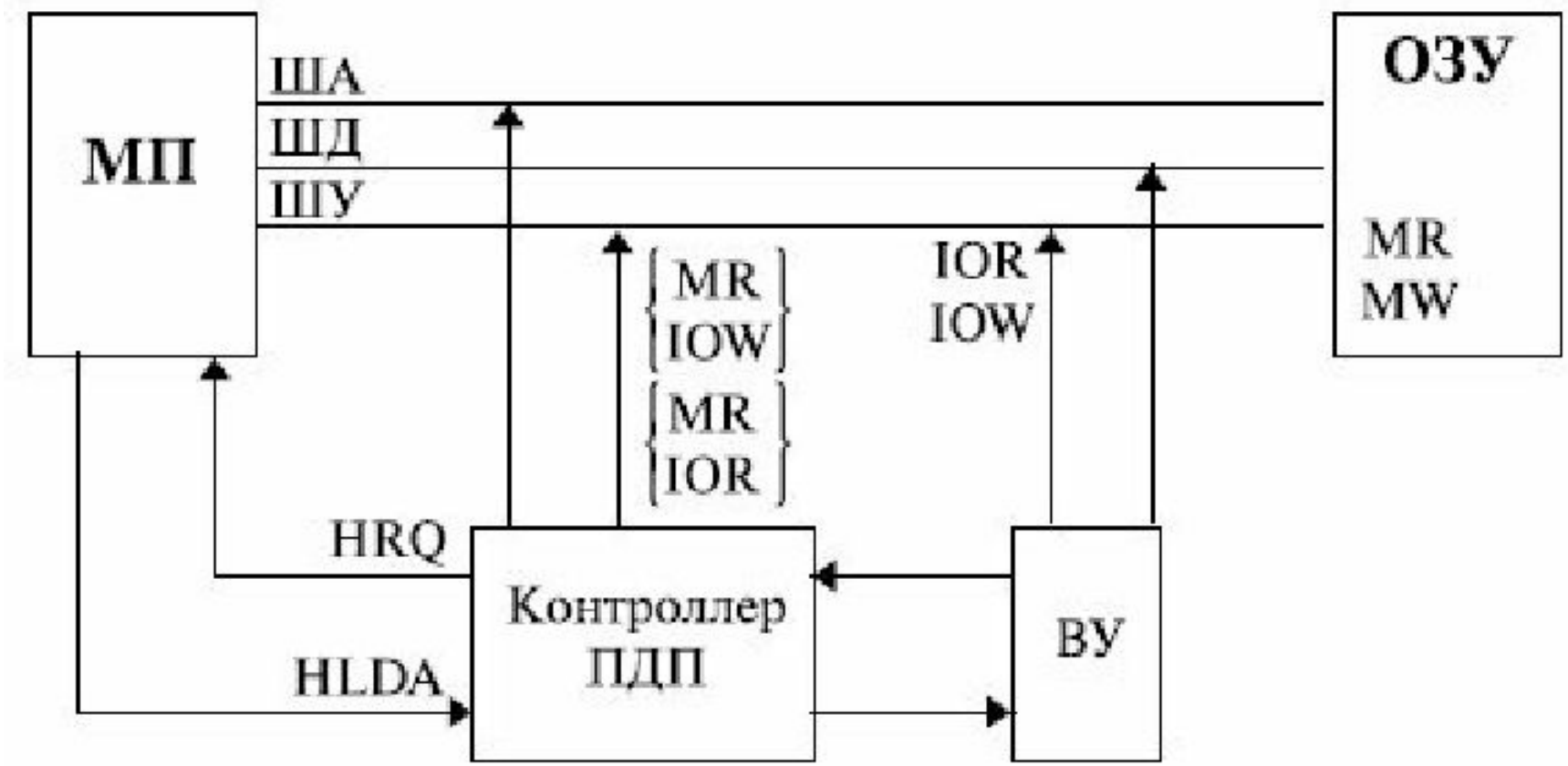
во внешнее устройство, адрес которого находится в регистре `DX` ; при этом вырабатывается сигнал IOW (`INput/OUTput Write`

1. с общим адресным пространством внешних устройств и оперативной памяти;
2. с независимыми адресными пространствами.

Программная реализация

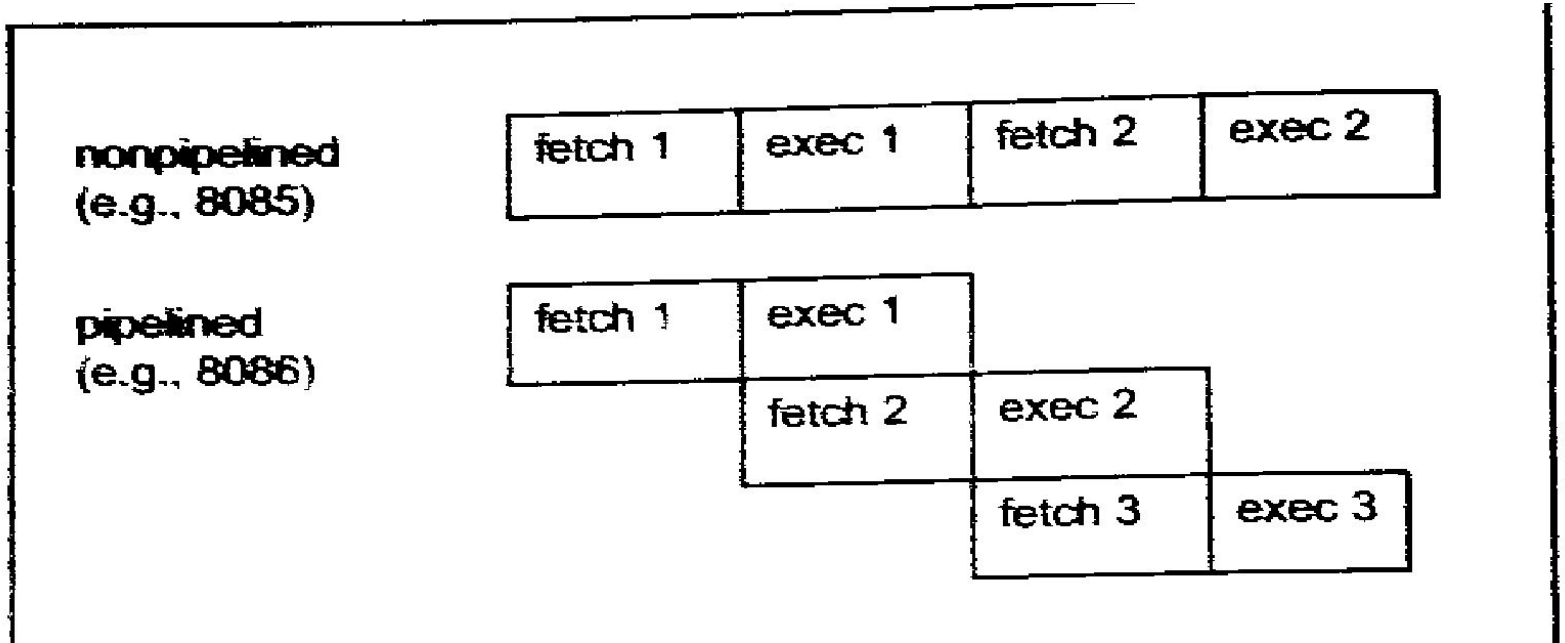
1. сформировать начальный адрес области обмена ОП;
2. занести длину передаваемого массива данных в один из своих внутренних регистров, который будет играть роль счетчика;
3. выдать команду чтения информации из ВУ; при этом на шину адреса из МП выдается адрес ВУ, на шину управления - сигнал чтения данных из ВУ, а считанные данные заносятся во внутренний регистр МП;
4. выдать команду записи информации в ОП; при этом на шину адреса из МП выдается адрес ячейки оперативной памяти, на шину управления - сигнал записи данных в ОП, а на шину данных выставляются данные из регистра МП, в который они были помещены при чтении из ВУ;
5. модифицировать регистр, содержащий адрес оперативной памяти;
6. уменьшить счетчик длины массива на длину переданных данных;
7. если переданы не все данные, то повторить шаги 3-6, в противном случае закончить обмен.

Аппаратная реализация (с ПДП)



Повышение производительности МП за счет параллелизма

Конве́йер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени - эксплуатация параллелизма на уровне инструкций



Конвейер команд

Команда	Такт								
	1	2	3	4	5	6	7	8	9
i	IF	ID	OR	EX	WB				
i+1		IF	ID	OR	EX	WB			
i+2			IF	ID	OR	EX	WB		
i+3				IF	ID	OR	EX	WB	
i+4					IF	ID	OR	EX	WB

ID (англ. *Instruction Decode*) — раскодирование инструкции,
EX (англ. *Execute*) — выполнение,
MEM (англ. *Memory access*) — доступ к памяти,
WB (англ. *Register write back*) — запись в регистр
OR (operands reading) - чтение операндов

Оценка эффективности

Количество команд	Время	
	при последовательном выполнении	при конвейерном выполнении
1	100	150
2	200	180
10	1000	420
100	10000	3120

микропроцессоре Pentium длина конвейера составляла 5 ступеней (при максимальной тактовой частоте 200 МГц), то в процессорах Pentium 4 на ядре Northwood длина конвейера составляла 20 ступеней, а на ядре Prescott она увеличена до 31 ступени при максимальной тактовой частоте 3,8 ГГц.

Конфликты

Конфликты делятся на три группы

- структурные,
- по управлению,
- по данным.

Структурные конфликты возникают в том случае, когда аппаратные средства процессора не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением.

1. Конфликты типа RAW (Read After Write - чтение после записи):
2. Конфликты типа WAR (Write After Read - запись после чтения):
3. Конфликты типа WAW (Write After Write - запись после записи):

Названия указывают должный порядок выполнения. Конфликты по управлению возникают при конвейерном выполнении условных передач управления и других команд, которые изменяют значение программного счетчика.