

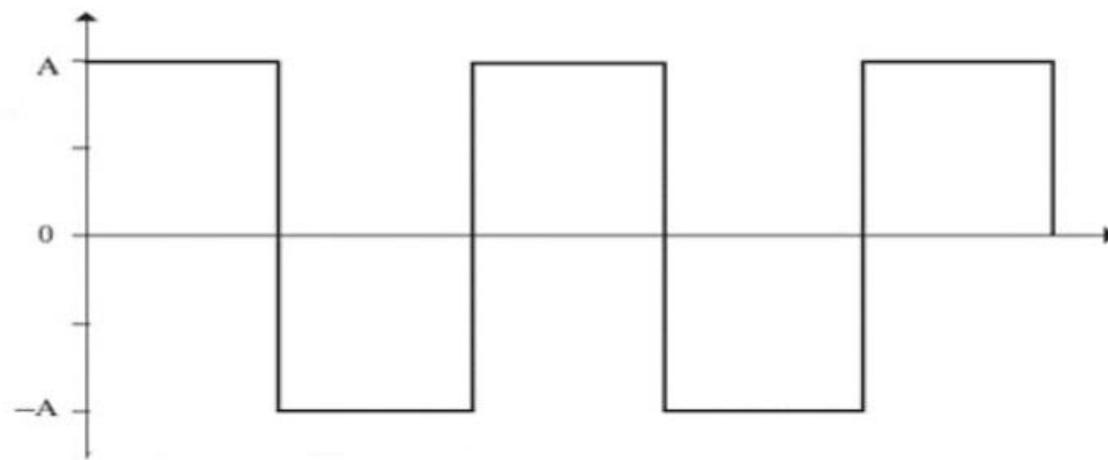
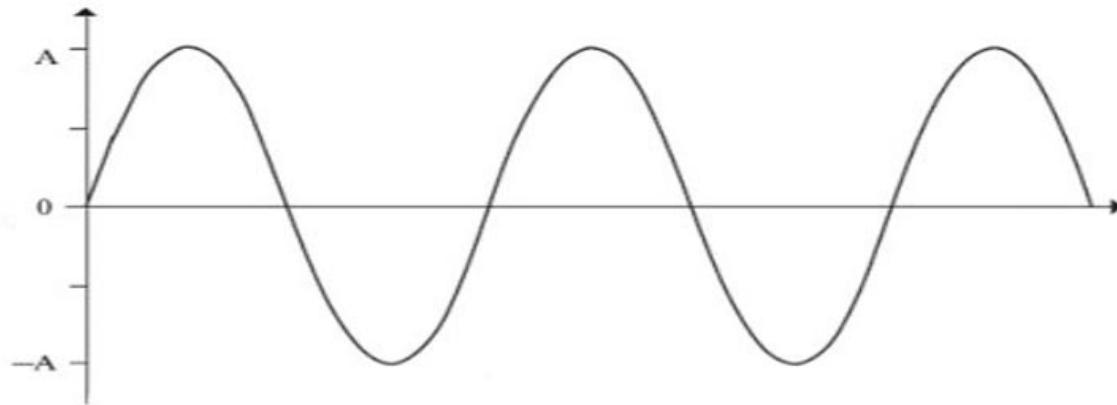
Цифровые схемы Процессоры Intel x86 Представление данных

Лекция №1 из курса АКОС

Электричество



Цифра v.s. аналог



Состояние выводов

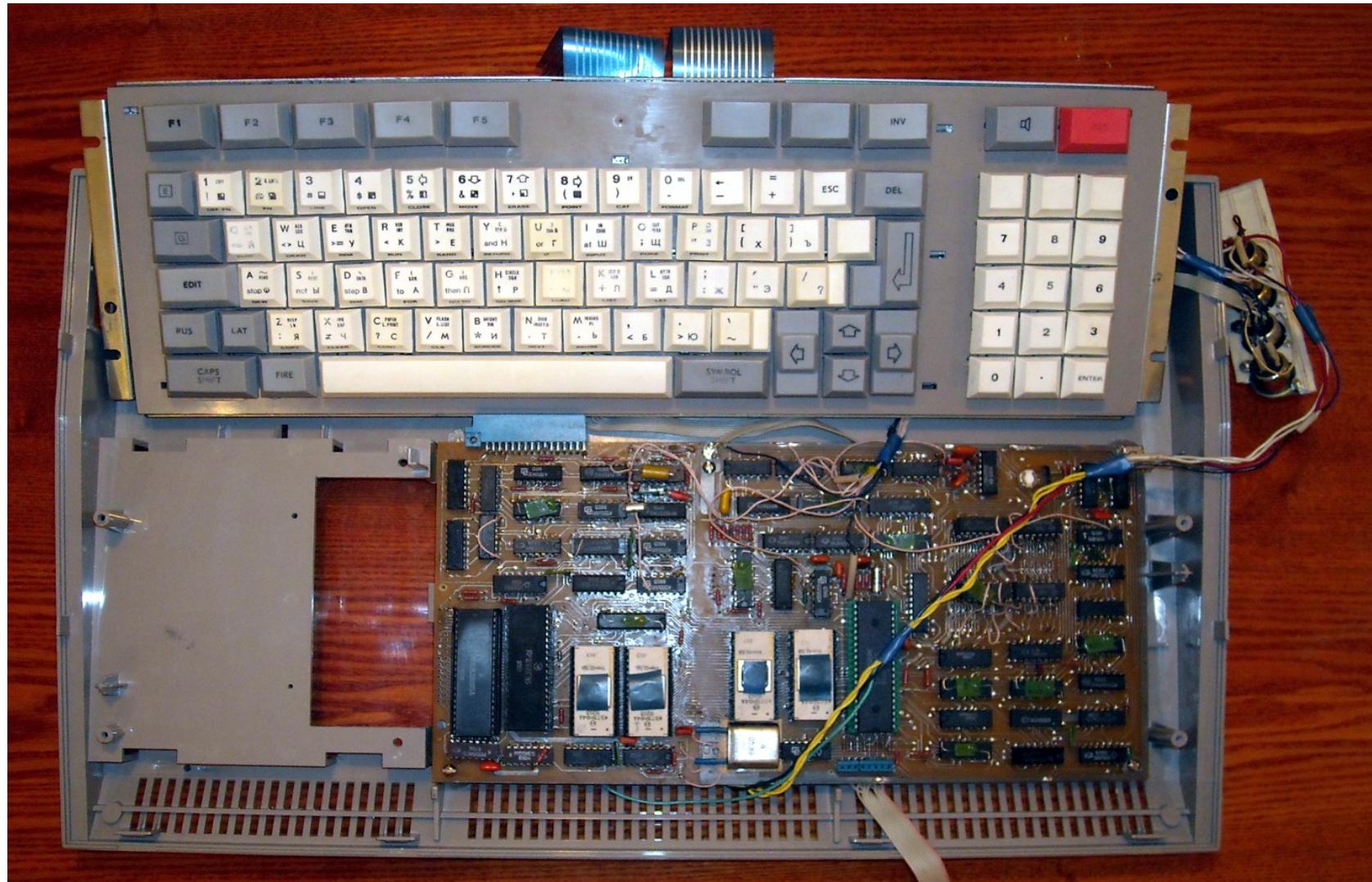
AD0	16
AD1	15
AD2	14
AD3	13
AD4	12
AD5	11
AD6	10
AD7	9
AD8	8
AD9	7
AD10	6
AD11	5
AD12	4
AD13	3
AD14	2
AD15	39
	--

- **Логический «0»** - это напряжение ниже порога нуля (0.7 вольт)
- **Логическая «1»** - напряжение выше порога единицы (около 2 вольт)
- Состояние между пороговыми значениями считается **неопределенным**
- **Высокоимпедансное состояние** - выводы отключены от логики микросхемы
- Некоторые сигналы инвертированы

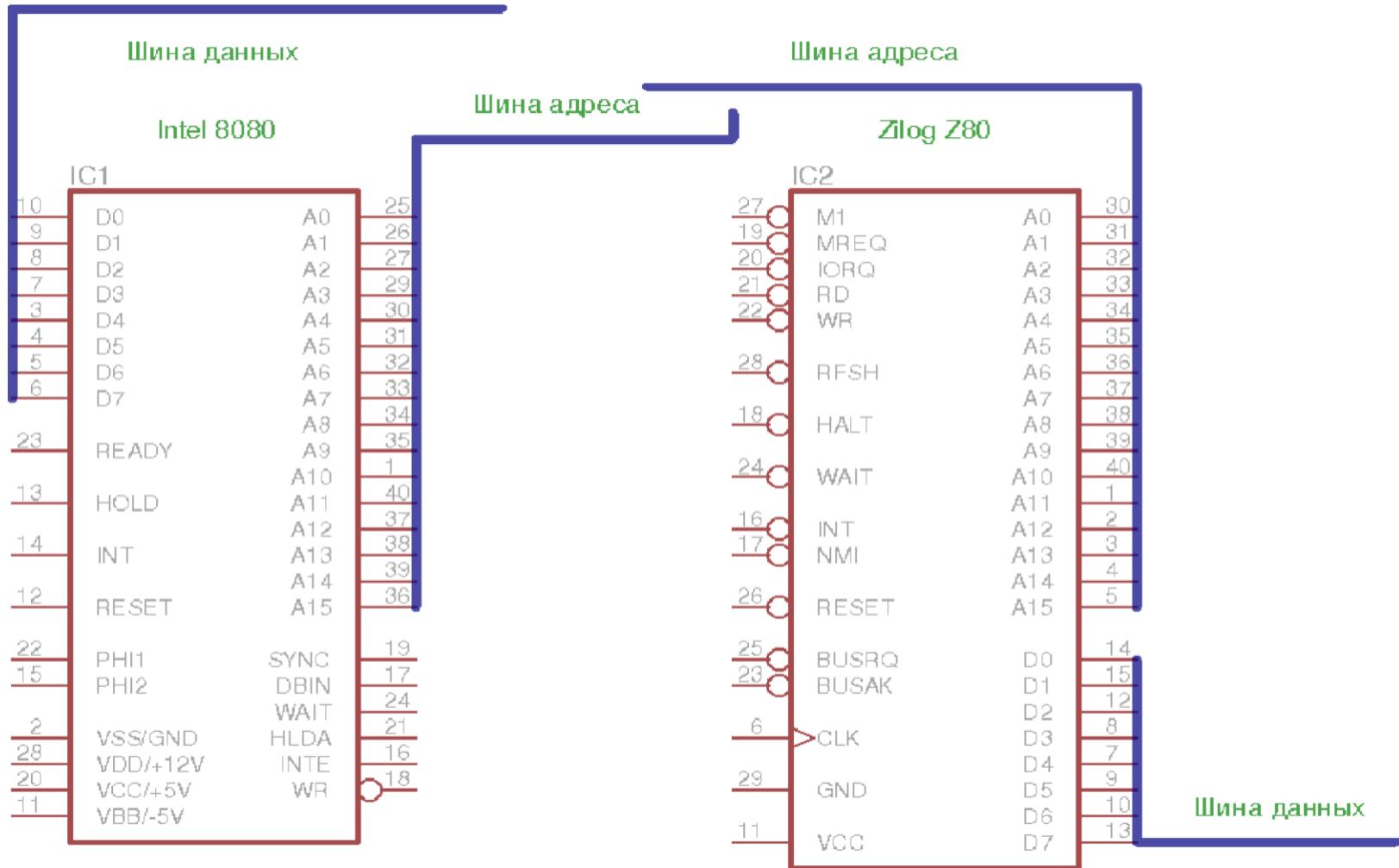
Устройство ПК



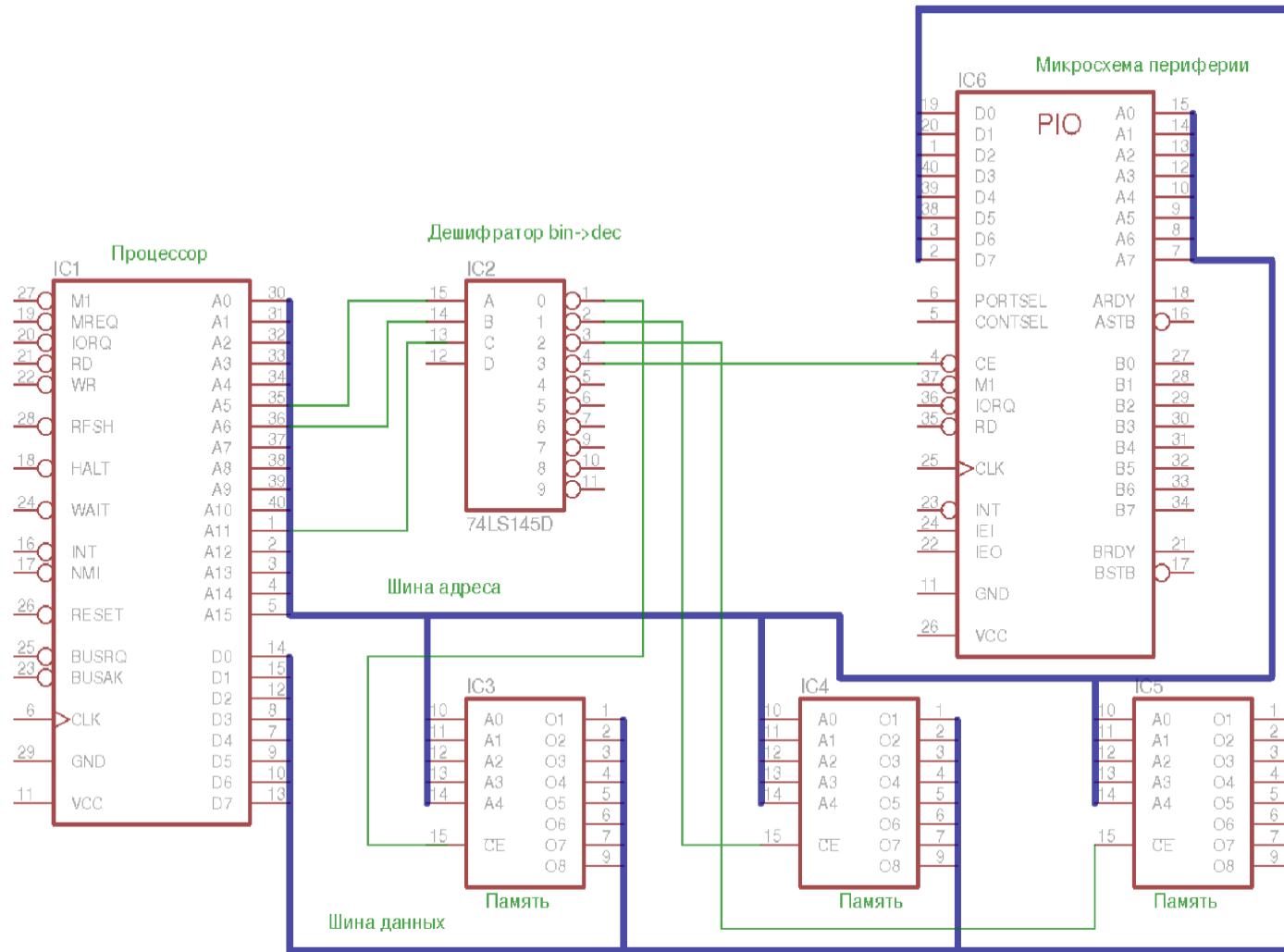
Устройство Ретро-ПК



Процессор 8080/Z80



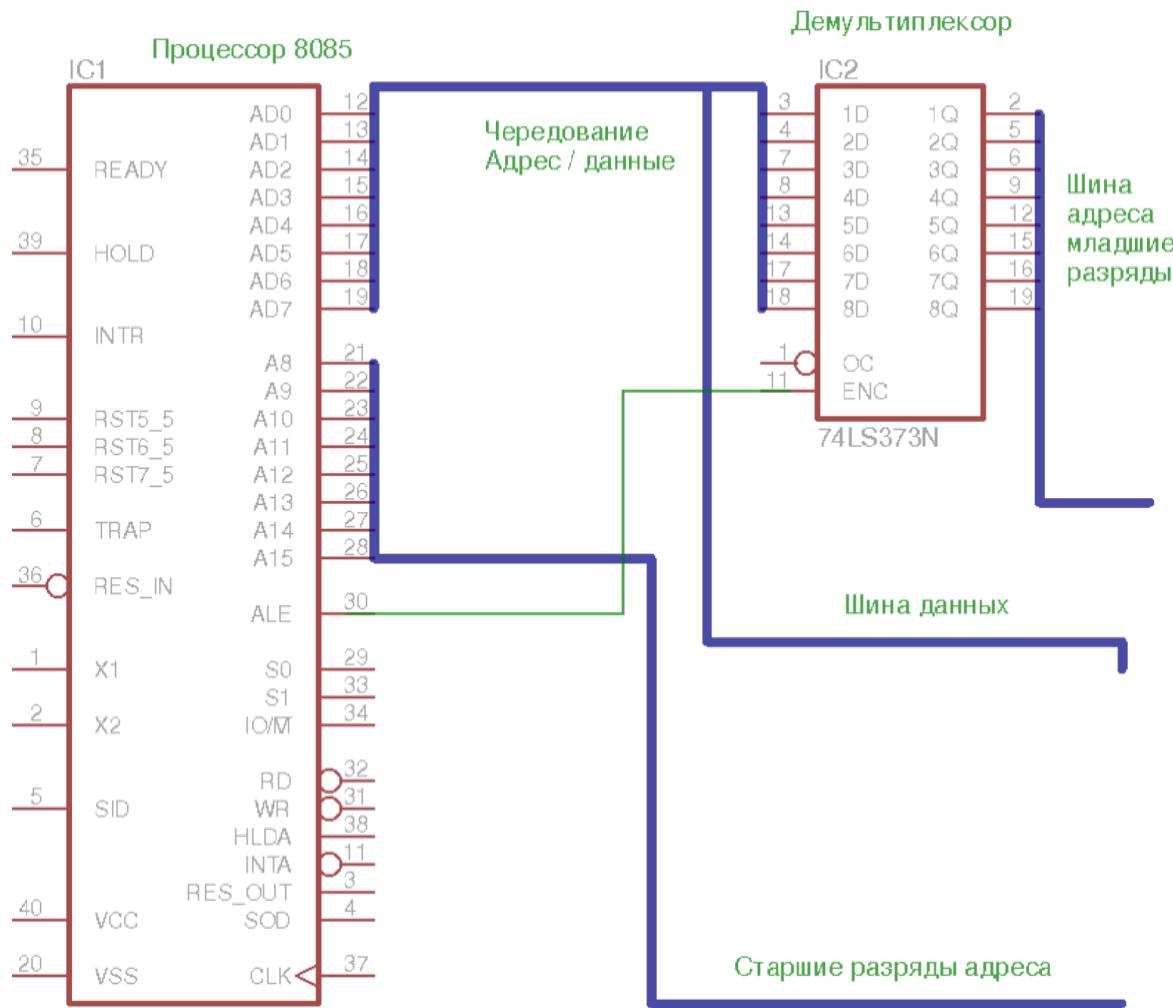
Шины адреса и данных



Архитектуры систем

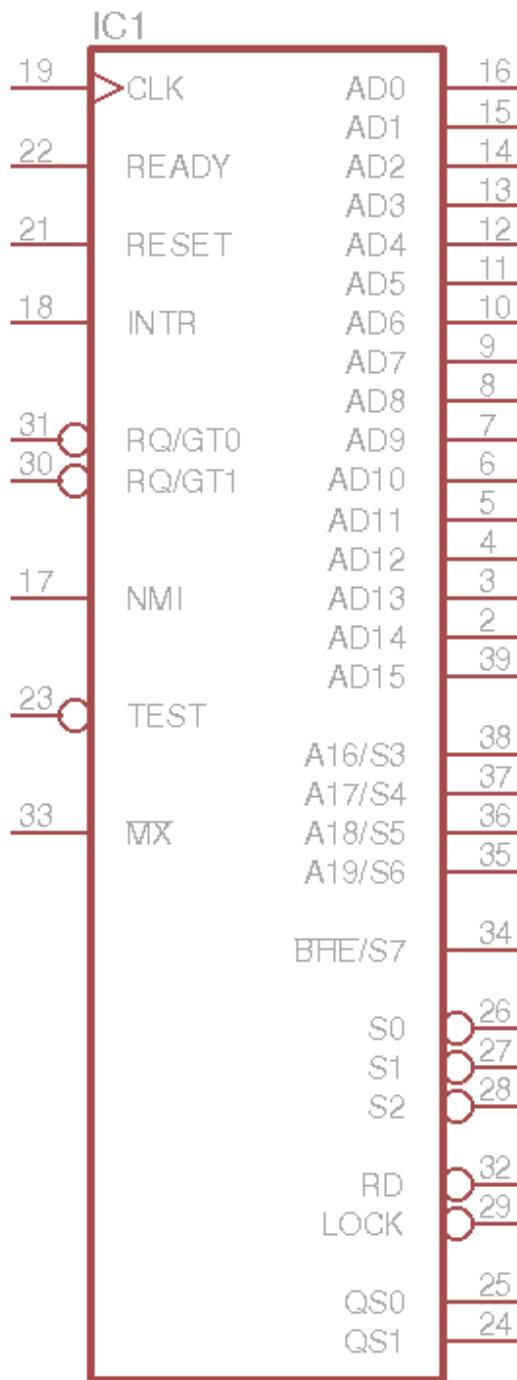
- **Архитектура Фон-Неймана**
Единое адресное пространство
для программ и данных
большинство процессоров
- **Гарвардская архитектура**
Раздельное адресное пространство
для программ и данных
микроконтроллеры

Шина адреса/данных



Семейство процессоров Intel

- 1974: Intel 8080/8085 / Zilog Z80 / ⚡ KP580ВМ80А
- 1978: Intel 8086/8088/80186 / ⚡ K1810ВМ88/K1810ВМ88
- 1982: Intel 80286 / AMD 80286 / ⚡ KP1847ВМ286
- 1985: Intel 80386 / AMD 386
- 1991: Intel 486 / AMD 486/5x86
- 1993: Intel P5 (поколение i586)
- 1996: Intel P6 (поколение i686)
- 2003: AMD K8 (архитектура x86_64)

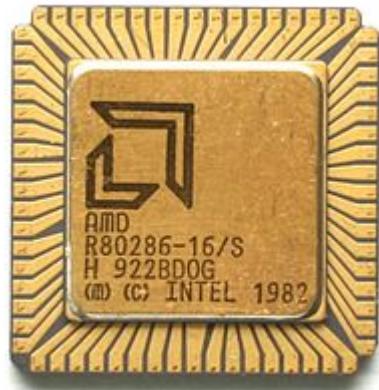


Intel 8086

- Смешанная 16-битная системная шина адреса/данных
- Всего 20 бит на адрес - сегментная модель памяти
- Коды команд отличаются от 8080/8085
- Программы для 8080/8085 могут работать после переассемблирования

**Все современные процессоры
x86 (IA-32, x86-64)
умеют выполнять программы
для Intel 8086**

Intel 80286



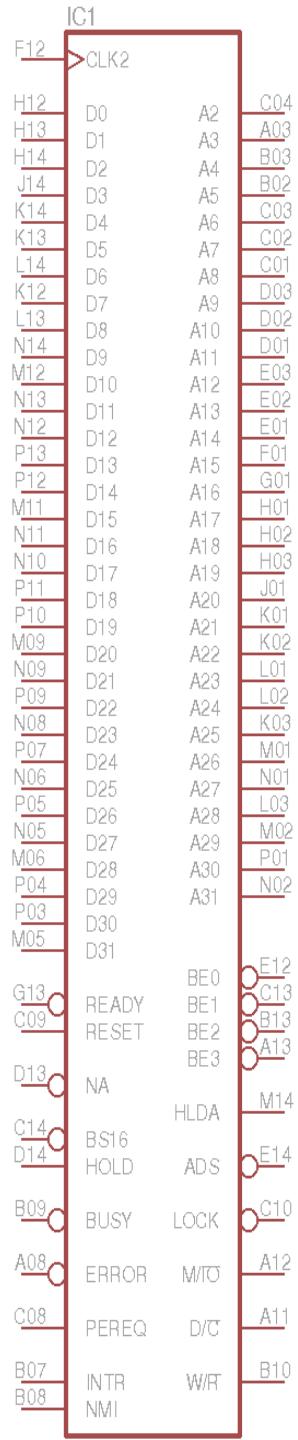
- Шина адреса 24 бит, шина данных 16 бит
- Раздельные шины адреса/данных
- Появился Memory Management Unit (MMU) - «зашщищенный режим»

IC1		
63	READY	A0
31	CLK	A1
29	RESET	A2
5	S0	A3
4	S1	A4
67	M/I/O	A5
68	LOCK	A6
59	NMI	A7
57	INTR	A8
64	HOLD	A9
65	HLDA	A10
53	ERROR	A11
54	BUSY	A12
61	PEREQ	A13
6	PEACK	A14
66	COD/INTA	A15
52	CAP	A16
		D0
		D1
		D2
		D3
		D4
		D5
		D6
		D7
		D8
		D9
		D10
		D11
		D12
		D13
		D14
		D15

Intel 80386



- Раздельные шины данных и адреса, каждая 32 бит.
- Размер указателя совпадает с размером типа `int`
- Обратно совместим с 8086 и 80286, отдельные команды для работы с 32-битными регистрами
- Поддержка страничной организации памяти в MMU



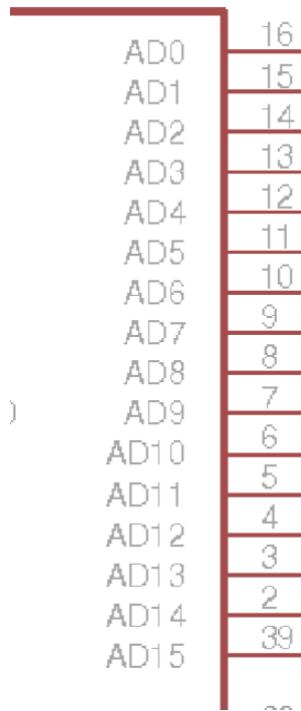
Intel 486 и Pentium P5



- Схемотехника - похожа на 386, различия только внутри
- Кэш-память на процессоре
- Внутреннее умножение тактовой частоты
- Инструкции для атомарных операций
- Конвейер(ы) для команд: один для 486, два для P5
- Векторные инструкции (Pentium MMX)

*Стали актуальны оптимизации
на уровне компилятора
компилятора*

Кодирование данных



- Минимально адресуемый объем памяти - один байт
- Процессор за один такт может прочитать только целиком одно **машинное слово**
- Внутри байта порядок бит определяется семантикой процессора
- Внутри машинного слова приняты разные порядки байт

Big Endian v.s. Little Endian

- BigEndian - естественный порядок записи байт: сначала старшие разряды, потом младшие. Принят в RISC-процессорах и является стандартом для передачи данных по сети
- LittleEndian - порядок байт в x86: сначала адресуются младшие байты, затем - старшие

Выравнивание в структурах

```
// i80386
typedef
struct two_fields {
    int16_t field_1; // 16 bit
    char     field_2; // 8 bit
} two_fields_t;

printf(«%d», sizeof(
            int_and_char_t
        ));
```

- Наиболее выгодный для x86 способ хранения данных - блоками, кратными машинному слову
- Поля в структурах прижимаются к границам

Представление данных

- Целые **беззнаковые** числа: всё очевидно
- Целые **знаковые** числа: по стандарту языка Си - формат не определен. Но де-факто все современные процессоры используют *обратный дополнительный код*

Беззнаковый эквивалент	Двоичное представление	Прямой код	Обратный дополнительный
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	-0	-8
9	1001	-1	-7
10	1010	-2	-6
11	1011	-3	-5
12	1100	-4	-4
13	1101	-5	-3
14	1110	-6	-2
15	1111	-7	-1

Целочисленное переполнение

$$X + Y = Z + C$$

Числа X , Y , Z - имеют фиксированную
разрядность

$C > 0$ -- это ситуация переполнения

Целочисленное переполнение

Дано: X, Y, Z - числа с фиксированной разрядностью

Теорема:

$$X + Y = Z + C$$

$C = 0$ тогда и только тогда,
когда $Z > X$ и $Z > Y$

Дополнительное условие: $X > 0$ и $Y > 0$, то есть, этот критерий работает только для **беззнаковых** чисел.

Целочисленное переполнение

- По стандарту языка Си (и многих других ЯП) нет явного способа проверить переполнение
- Но многие процессоры эту ситуацию отлавливают
- Use GCC builtins

Представление строк

- Стандарт ASCII - 7 бит на базовую латиницу
- Коды [128,255] - для национальных символов. Код -1 = 255 - русская буква «я» в кодировке Windows-1251 или «ъ» в кодировке KOI8-R
- Универсальная кодировка UTF-8 с переменным числом байт
- 2 или 4 байта на символ для Unicode

man 7 utf8

0x00000000 - 0x0000007F:

0xxxxxxx

0x00000080 - 0x000007FF:

110xxxxx 10xxxxxx

0x00000800 - 0x0000FFFF:

1110xxxx 10xxxxxx 10xxxxxx

0x00010000 - 0x001FFFFFF:

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

0x00200000 - 0x03FFFFFF:

111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

0x04000000 - 0x7FFFFFFF:

1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

wchar_t v.s. utf8

- `sizeof(wchar_t)==4` (в Unix) или 2 (MSVC)
- использование `wchar_t` гарантирует корректность простой индексации оператором `[]`
- кодировка utf8 удобна только для последовательной обработки или хранения символов