

Архитектура ЭВМ

К лабе 0

Архитектура фон Неймана

- Принцип **двоичного кодирования данных**
- Принцип **однородности памяти**
 - Код программы и ее данные хранятся в одной и той же памяти. Поэтому ЭВМ не различает, что хранится в данной ячейке памяти — число, текст или команда. Над командами можно выполнять такие же действия, как и над данными.
- Принцип **адресуемости памяти**
 - Основная память структурно состоит из линейно пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка -> Возможность использования переменных.
- Принцип **последовательного программного управления**
 - Предполагает, что программа состоит из набора команд, которые выполняются процессором автоматически друг за другом в определенной последовательности. Для изменения прямолинейного хода необходимы специальные команды перехода.
- Принцип **жесткости архитектуры**
 - Неизменяемость в процессе работы топологии, архитектуры, списка команд.

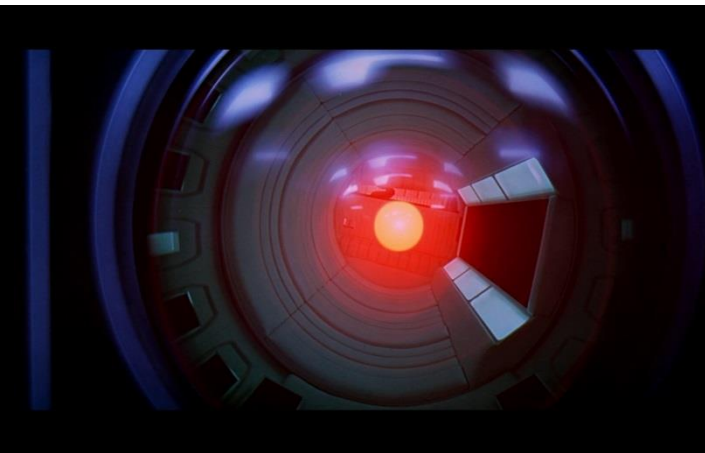
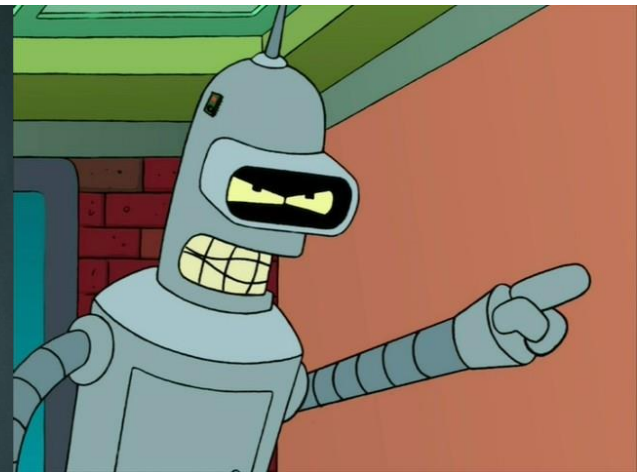
ЭВМ



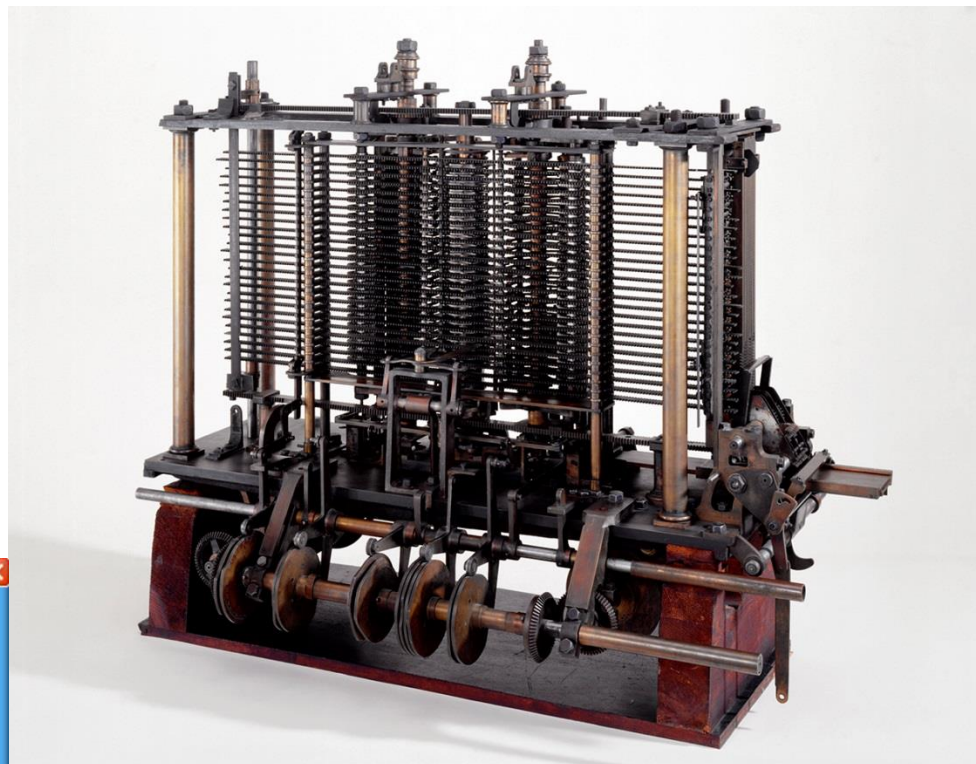
НИКС.RU
WWW.NIKS.RU



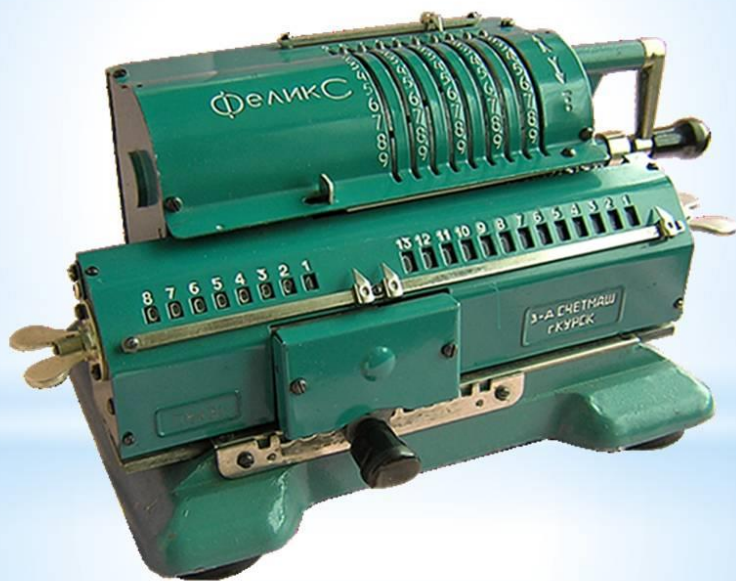
ЭВМ



Механические счетные машины



Арифмометр "Феликс", 1950-е годы (СССР).



Программируемый калькулятор

```
GO TO (())  
SUB  
0  
8      Call the subroutine at line 08 to calculate the area  
x      Multiply by the length in y  
STOP    Display it  
GO TO (()) This shouldn't be an END!  
0  
0      Ready for the next case  
ENTER^  This key is labeled with an up arrow  
x      Square the radius  
PI      Multiply by PI  
x      (Now have X=PI, Y=area, Z=length)  
roll dn  Move area to X, length to Y  
RETURN  
END
```



Hewlett-Packard 9100A


```
BB 11 01 B9 0D 00 B4 0E 8A 07 43 CD 10 E2 F9 CD 20 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21
```

Двумерный битовый массив или массив символов

111 1980 T. TV 23-00-1-79

Пример кода:

```

/&-0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ:#@'=".[<(+|]$*);^\\,%_~?
12 / X          XXXXXXXXXX          XXXXXX
11|  X          XXXXXXXXXX          XXXXXX
0|   X          XXXXXXXXXX          XXXXXX
1|    X          X          X          XXXXXXXXXX          XXXXXX
2|     X          X          X          X          X          X          X          X
3|      X          X          X          X          X          X          X          X
4|       X          X          X          X          X          X          X          X
5|        X          X          X          X          X          X          X          X
6|         X          X          X          X          X          X          X          X
7|          X          X          X          X          X          X          X          X
8|           X          X          X          X          X XXXXXXXXXXXXXXXXXXXXXXXXXXXX
9|            X          X          X          X

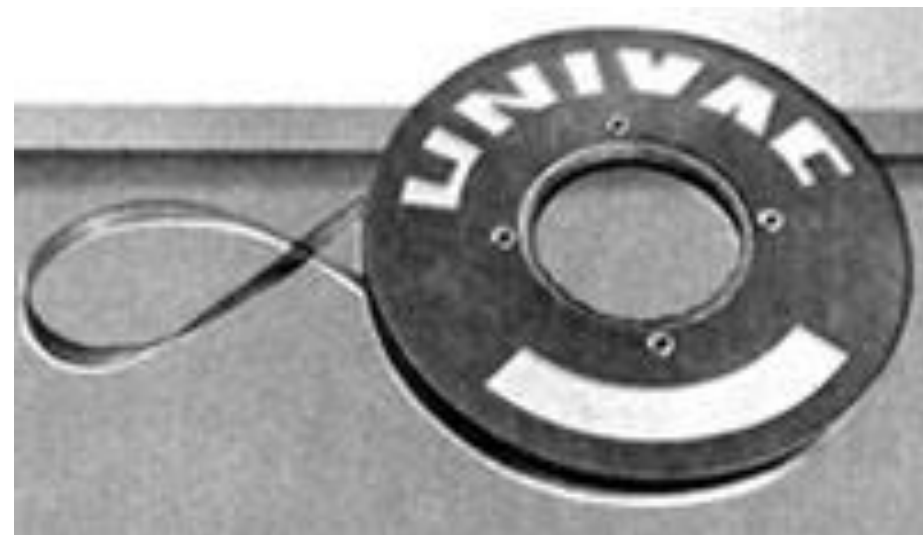
```

Компьютер UNIVAC

первый компилятор

Even by today's standards the UNIVAC I instruction set was quite advanced. Its 43 instructions are shown.

- Twelve (B C F G H J K L V W Y Z) for loading, preparing, and storing registers.
- Seven (A D M N P S X) for arithmetic.
- Two (T Q) for conditional transfers.
- One (U) for unconditional transfer.
- One (R) for subroutine return.
- Four (. : - 0) for shifts.
- One (E) for masking.
- One (00) for skipping.
- One (90) for stopping.
- One (10) for typing one word into long term memory from the SC keyboard.
- One (50) for typing one word from long term memory to the SC on-line typewriter.
- One (,) for a breakpoint stop.
- Ten (1 2 3 4 5 6 7 8 30 40) for input/output, including backward reading from UNISERVOs.



Материнская плата



Main Logic

- 1. CPU socket
- 2. Chipset Northbridge
- 3. Chipset Southbridge

Memory

- 4. DRAM Channel 1
- 5. DRAM Channel 2

Drive Interfaces

- 6. Floppy Drive
- 7. ATA100/ATA133
- 8. Serial ATA

Expansion Slots

- 9. PCI (32-bit, 33MHz)
- 10. PCI-Express x16
- 11. PCI-Express x1

Power Connectors

- 12. 24-pin ATX Power
- 13. 8-pin ATX12v Power
- 14. Supplemental Graphics Power

Onboard Features

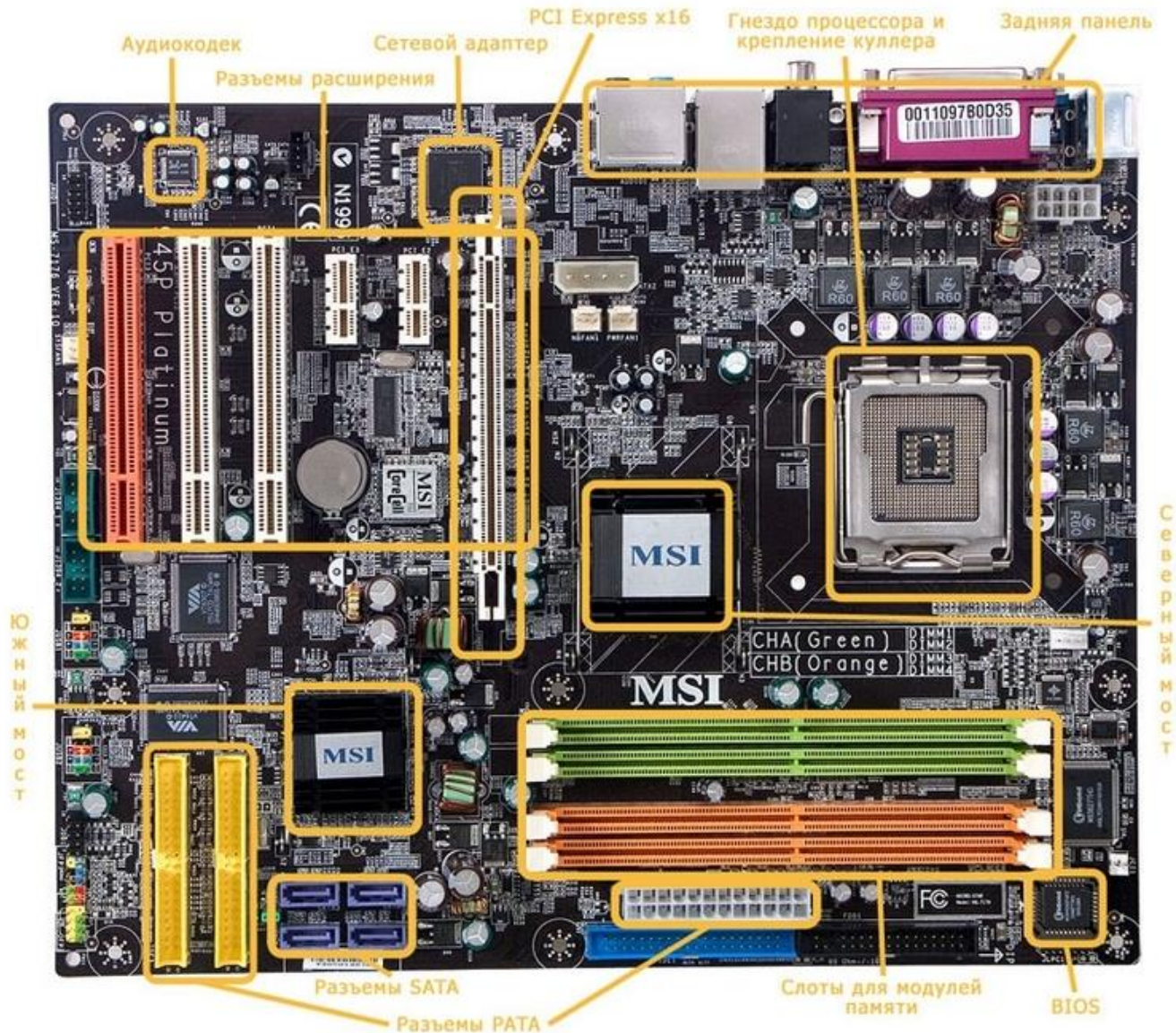
- 15. CPU Power Regulators
- 16. IEEE1384 FireWire Controller
- 17. Audio Codec
- 18. Network Controller PHY

BIOS

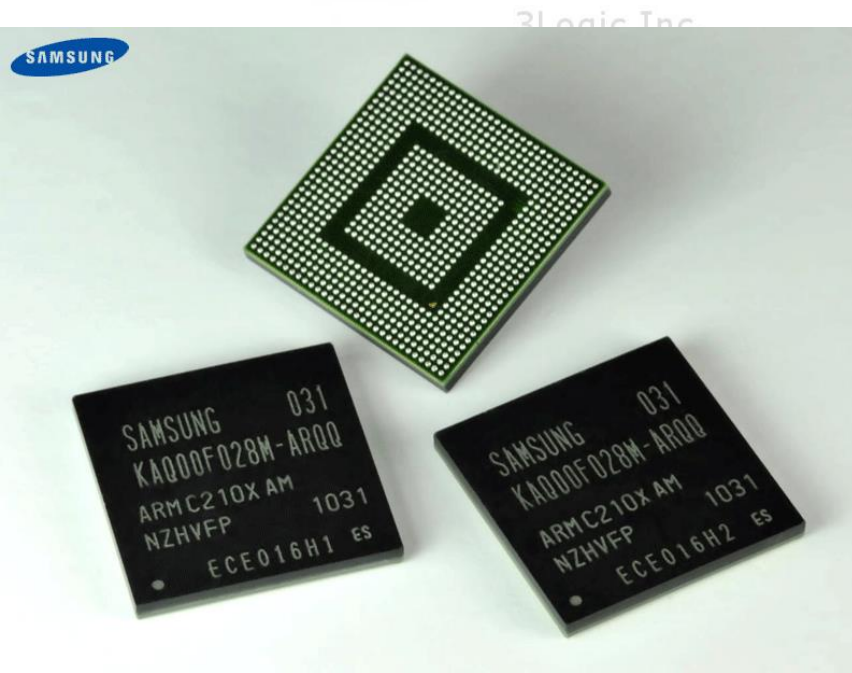
- 19. BIOS ROM (CMOS)
- 20. BIOS Clock Battery

О сбое компьютер вы узнаете по яркой вспышке, клубам дыма, фонтану искр и взрыву, который отбросит вас от компьютера на несколько метров
(с) классика Голливуда

Материнская плата



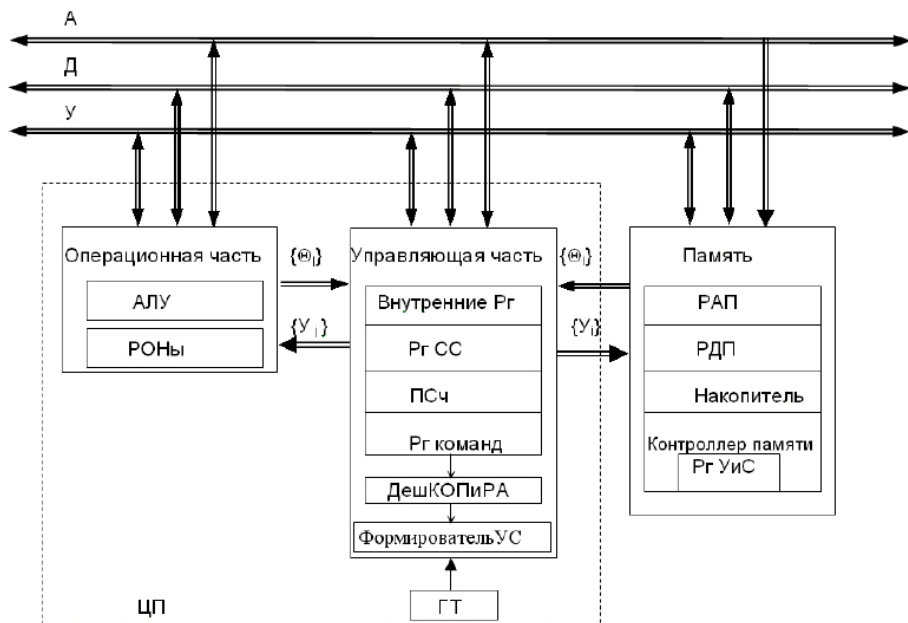
Процессор



Intel, AMD, IBM

ARM: Apple, Nvidia,
Samsung, Qualcomm, Sony
Ericsson, Texas Instruments,
Broadcom, HiSilicon
Technologies

Процессор



1. Выборка команды (I F).
2. Формирование исполнительных адресов операндов, если требуется (A M).
3. Выборка операндов из памяти (O F).
4. Исполнение операции (E X).
5. Запоминание результата (S T).
6. Проверка запроса программного прерывания (I R Q).

- АЛУ – арифметико-логическое устройство выполняет операции по обработке данных;
- РОНЫ – регистры общего назначения (от 8 до нескольких сотен штук) – сверхбыстрая память малой емкости для хранения операндов;
- Рг СС – регистр слова состояния. Содержит текущее состояние процессора;
- ПСч – программный счетчик. Содержит адрес текущей команды и автоматически наращивается для подготовки адреса следующей команды (исключение составляет команда перехода);
- Рг Команд – регистр команд. Содержит код исполняемой в данный момент команды;
- ДешКОПиРА – дешифратор кода операции и режимов адресации;
- Формирователь УС – формирователь управляющих сигналов $\{U_i\}$;
- РАП - регистр адреса памяти; РДП - регистр данных памяти;
- Рг УиС – регистр управления и состояния контроллера памяти.

Машинный код

Программа «Hello, world!» [\[править | править вики-текст \]](#)

Программа «Hello, world!» для процессора архитектуры x86 (ОС MS DOS, вывод при помощи BIOS прерывания int 10h) выглядит следующим образом (в шестнадцатеричном представлении):

BB 11 01 B9 0D 00 B4 0E 8A 07 43 CD 10 E2 F9 CD 20 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21

Комментарии к программе

[\[скрыть\]](#)

Данная программа работает при её размещении по смещению 100_{16} . Отдельные инструкции выделены цветом:

- BB 11 01, B9 0D 00, B4 0E, 8A 07 — команды присвоения значений регистрам.
- 43 — инкремент регистра BX.
- CD 10, CD 20 — вызов программных прерываний 10_{16} и 20_{16} .
- E2 F9 — команда для организации цикла.
- Малиновым показаны данные (строка «Hello, world!»).

Тот же код ассемблерными командами:

Код Ассемблера

Примеры программы [Hello, world!](#) для разных платформ и разных диалектов:

Пример [COM](#)-программы для [MS-DOS](#) на диалекте [TASM](#)

[\[показать\]](#)

Пример [EXE](#)-программы для [MS-DOS](#) на диалекте [TASM](#)

[\[показать\]](#)

Пример программы для [Linux/x86](#) на диалекте [NASM](#)

[\[скрыть\]](#)

```
SECTION .data
msg: db "Hello, world",10
len: equ $-msg

SECTION .text
global _start
_start: mov edx, len
        mov ecx, msg
        mov ebx, 1    ; stdout
        mov eax, 4    ; write(2)
        int 0x80

        mov ebx, 0
        mov eax, 1    ; exit(2)
        int 0x80
```

Соответствие

```
XXXX:0100    mov    bx, 0111h    ; поместить в bx смещение строки HW
XXXX:0103    mov    cx, 000Dh    ; поместить в cx длину строки HW
XXXX:0106    mov    ah, 0Eh      ; поместить в ah номер функции прерывания 10h
XXXX:0108    mov    al, [bx]      ; поместить в al значение ячейки памяти, адрес
которой находится в bx
XXXX:010A    inc    bx           ; перейти к следующему байту строки (увеличить
смещение на 1)
XXXX:010B    int     10h         ; вызов прерывания 10h
XXXX:010D    loop   0108        ; если cx≠0, то уменьшить cx на 1 и перейти по
адресу 0108
XXXX:010F    int     20h         ; прерывание 20h: завершить программу
XXXX:0111 HW db    'Hello, World!' ; строка, которую требуется напечатать
```

Ассемблер

Язык ассемблера позволяет понимать поведение машины:

– Настройка производительности программы

- Ассемблерные вставки
- Оптимальная структура кода

– Отладка ошибок

- Компиляторы создаются людьми

– Зловредный код

- Средства защиты
- Вирусы, трояны, руткиты ...

– Новое «железо»

Исключается для...

- функционально сложного ПО
- сопровождаемого ПО
- переносимого ПО

GCC и Ассемблер

```
int x=5, y=6, z=0;
int main()
{
    i=x+y;
    return 0;
}
```



GCC и Ассемблер

mkdir your_surname

<- создать папку

cd ./your_surname

<- перейти в папку

gedit as_test.c

<- текстовый
редактор

gcc as_test.c **-o** as_test.out

<- компиляция в
исполняемый файл

./as_test.out

<- запуск программы

gcc as_test.c **-S** **-o** as_test.s

<- ассемблерный листинг

gedit as_test.s

<- текстовый
редактор

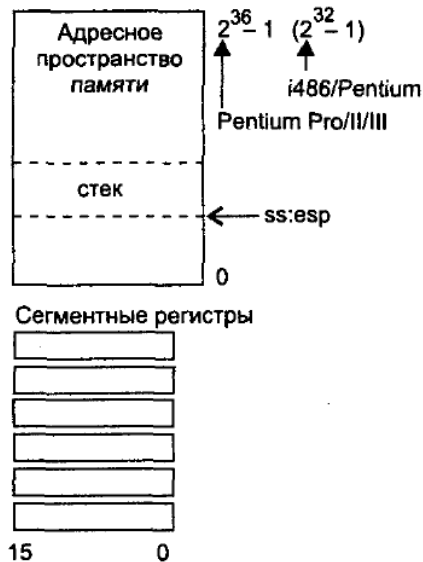
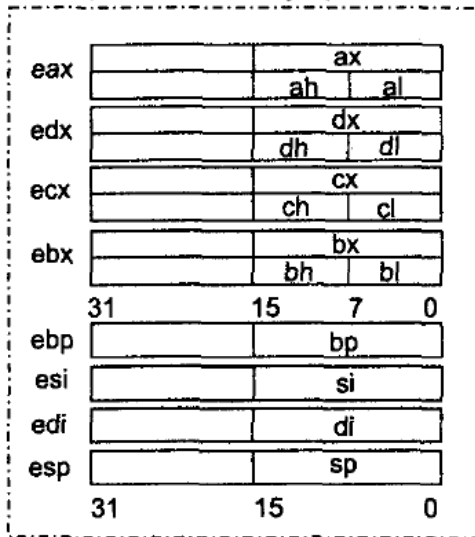
X86: Intel vs AT&T

[Intel 8086](#) — 16 бит;
[Intel 80186](#) — 16 бит;
[Intel 80286](#) — 16 бит;
[Intel 80386](#) — 32 бита;
[Intel 80486](#) — 32 бита;
Pentium — 32 бита;
Celeron — 32 бита;
Intel Core — 32 бита;
[Itanium](#) — 64 бита;
[Xeon](#) — 64 бита.

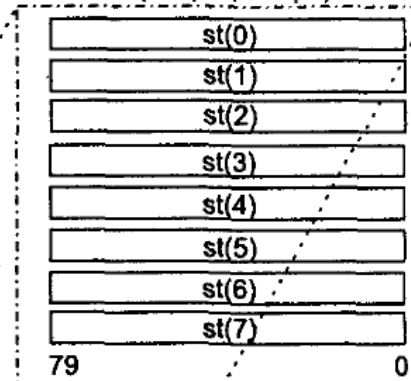
Intel Code		AT&T Code	
mov	eax, 1	movl	\$1, %eax
mov	ebx, 0ffh	movl	\$0xff, %ebx
mov	ebx, eax	movl	%eax, %ebx
mov	eax, [ecx]	movl	(%ecx), %eax
mov	eax, [ebx+3]	movl	3(%ebx), %eax
mov	eax, [ebx+20h]	movl	0x20(%ebx), %eax
add	eax, [ebx+ecx*2h]	addl	(%ebx, %ecx, 0x2), %eax
sub	eax, [ebx+ecx*4h-20h]	subl	-0x20(%ebx, %ecx, 0x4), %eax
lea	eax, [ebx+ecx]	leal	(%ebx, %ecx), %eax

- имена регистров начинаются с символа %
- размер операнда определяется как суффикс имени инструкции
movq %rax, %rbx movl %ebx, %eax movw %ax, %bx movb %al, %ah
- порядок операндов — вначале источник, затем приёмник
- числовые константы начинаются с символа \$
- мнемоники некоторых команд (например, cdq называется cld)
- команды ассемблера (такие, как объявление констант, резервирование места)
- отсутствие префикса операнда указывает на адрес в памяти

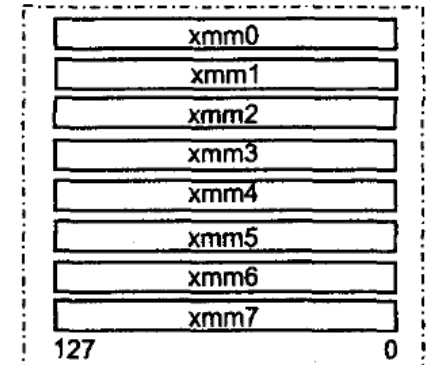
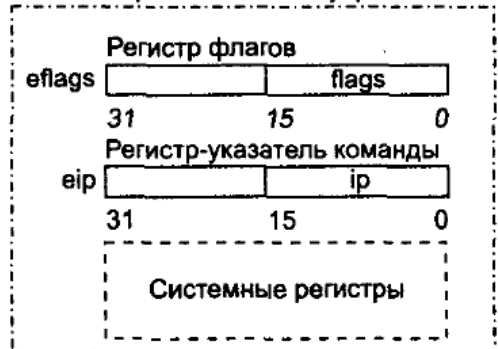
Регистры



Регистры устройства
с плавающей точкой
(сопроцессора)



Регистры состояния и управления



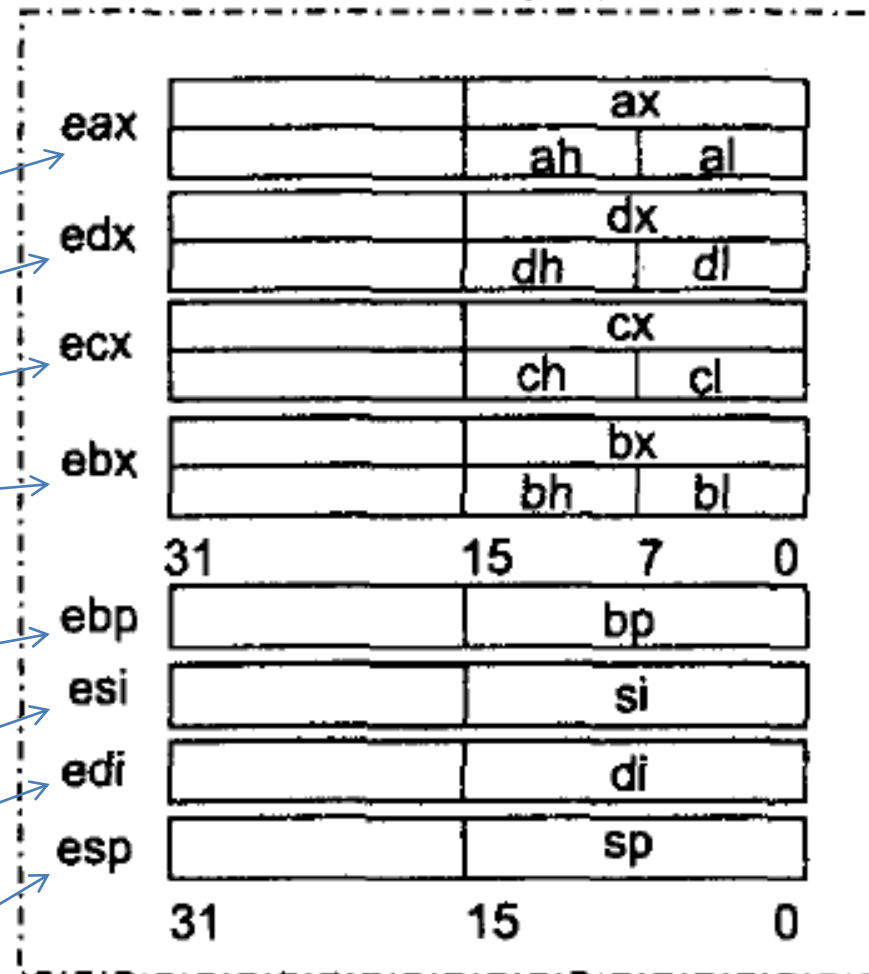
Регистры MMX-расширения с плавающей точкой (Pentium III)

Регистры общего назначения

- операнды логических и арифметических операций
- компоненты адреса
- указатели на ячейки памяти


- Аккумулятор
- Регистр данных
- Регистр-счетчик
- Базовый регистр
- Регистр указателя базы
кадры стека
- Индекс источника
- Индекс приемника
- Регистр указателя стека

Регистры общего назначения
целочисленного устройства



Арифметические операции

inc *операнд*
dec *операнд*
add *источник, приёмник*
sub *источник, приёмник*
mul *множитель_1*



Команда	Второй сомножитель	Результат
mulb	%al	16 бит: %ax
mulw	%ax	32 бита: младшая часть в %ax, старшая в %dx
mull	%eax	64 бита: младшая часть в %eax, старшая в %edx

GCC Inline Assembly

```
#include <stdio.h>
```

```
int x;  
int main()  
{  
    x=0;  
    x += 10;  
    printf("%d", x);  
    return 0;  
}
```

```
#include <stdio.h>
```

```
int x;  
int main()  
{  
    x=0;  
    asm("addl $10, x");  
    printf("%d", x);  
    return 0;  
}
```

```
#include <stdio.h>
```

```
int x;  
int main()  
{  
    asm(  
        "mark: ""movl $0, %eax\n"  
        "addl $10, %eax\n"  
        "movl %eax, x"  
    );  
    printf("%d", x);  
    return 0;  
}
```