

Архитектура компьютера

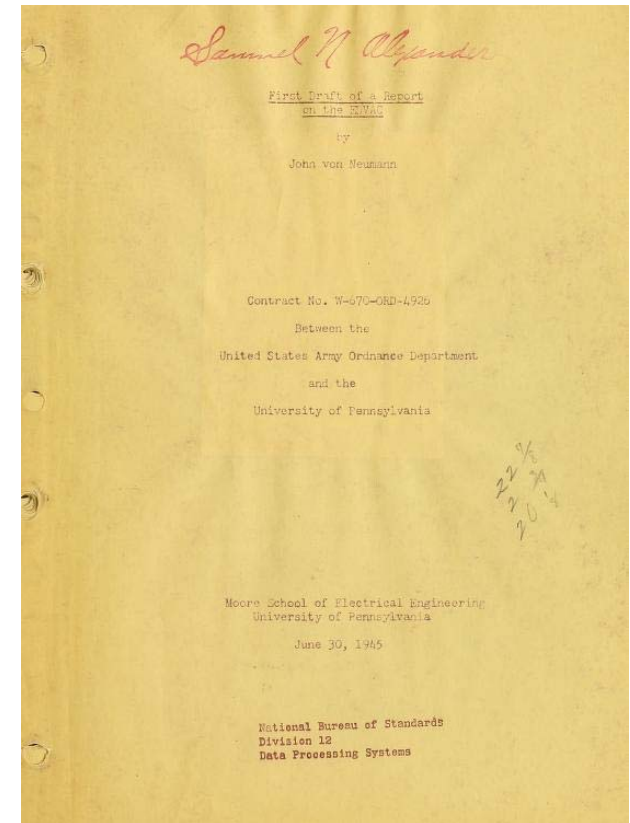
ЛЕКЦИЯ 2. МНОГОУРОВНЕВАЯ КОМПЬЮТЕРНАЯ ОРГАНИЗАЦИЯ

Классическая структура ЭВМ

30 июня 1945 года – «Первый проект отчёта о EDVAC» (First Draft of a Report on the EDVAC) — незаконченный 101-страничный документ, написанный Джоном фон Нейманом и распространённый Германом Голдстейном, куратором со стороны Армии США проектов ENIAC и EDVAC.

В документе впервые опубликовано описание логического устройства вычислительной машины с хранимой в памяти программой. Эта концепция в дальнейшем стала известна как **«архитектура фон Неймана»**.

Авторы ключевой идеи *хранимой в памяти программы* – весь коллектив создателей **ENIAC** и **EDVAC**: Джон Преспер Экерт, Джон Уильям Мокли, Герман Голдстейн, Джон фон Нейман и др.



Принципы фон Неймана

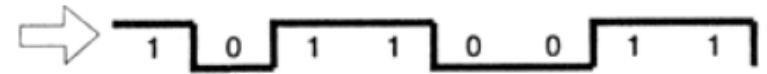
Основополагающие свойства архитектуры машины Фон Неймана формулированы в виде **принципов Фон Неймана**. Эти принципы в значительной степени определяли основные черты архитектуры двух первых поколений ЭВМ и оказали серьезное влияние на развитие компьютерной техники.

- **Принцип двоичного кодирования.**
- **Принцип однородности памяти.**
- **Принцип адресуемости памяти.**
- **Принцип последовательного программного управления.**
- **Принцип жесткости архитектуры.**

Принципы фон Неймана

1. Принцип двоичного кодирования.

Согласно этому принципу, вся информация, как данные, так и команды, кодируются двоичными цифрами 0 и 1.



Выбор двоичной системы обусловлен:

- Простотой технической реализации
- Согласованностью с булевой логикой
- Простотой выполнения арифметических и логических операций

Принципы фон Неймана

Недостатки классической двоичной системы счисления

«Проблема представления отрицательных чисел». Отрицательные числа непосредственно не могут быть представлены в классической двоичной системе счисления, использующей только две двоичные цифры 0 и 1.

«Нулевая избыточность». В процессе передачи, хранения или обработки двоичной кодовой комбинации, например 10011010, под влиянием "помех", действующих в "канале", может произойти искажение данной кодовой комбинации и она перейдет в кодовую комбинацию 11010010, но, поскольку эта комбинация является "разрешенной", то не существует способа обнаружить данную ошибку.

Принципы фон Неймана

2. Принцип адресуемости памяти.

Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка.

```
> dump $2000 $2200
03-2000: 4083 130D 1311 1314 021F 2023 4C49 5354 @..... #LIST
03-2008: 204B 4559 A807 6C0F 0000 0828 47D1 443A KEY...1...<G.D:
03-2010: 44E0 0360 F187 607C 2034 321D 07EC 0A1D D...`...! 42....
03-2018: 47D7 F082 44F6 6C05 6D2D 4526 E8C6 F081 G...D.l.m-E&....
03-2020: 4523 F081 F081 2060 4C49 5354 A805 6C2C E#.... 'LIST..l,
03-2028: 0000 0A34 C12C 082A 47B3 441C 7171 834B ...4...*G.D.qq.K
03-2030: 2025 B800 7171 834B 2026 321D 0001 B21D %..qq.K &2....
03-2038: 834B 2025 8000 F020 2001 F403 47B8 6FBC .K %:... ..G.o.
03-2040: 0360 F187 607C 2038 321D 07BB 0A1D 47A8 `...! 82.....G.
03-2048: F082 7170 3360 7171 834B 2023 B800 7171 ..qp3`qq.K #..qq
03-2050: 2021 B800 F081 44BD 6C09 6D03 C103 E8C6 ?.....D.l.m....
03-2058: 4504 6D01 C103 E8C6 F081 C103 6CFA 6FP9 E.m.....l.o.
03-2060: 2077 5255 4E20 A805 6C0F 0000 4406 F081 wRUN ..l...D...
03-2068: 0796 086C 4785 F082 0824 4772 7171 834B ...lG....$Grqq.K
03-2070: 2023 B800 F082 C0CE C103 6CE3 F081 208A #.....l....
03-2078: 434F 4E54 A805 6C08 0000 47EF F081 077E CONT..l...G....~
03-2080: 086C 476E F082 C0CE C103 6C02 F081 0887 .lGn.....l....
03-2088: C0D3 F081 2097 5343 5241 5443 4820 4120 .... SCRATCH A
03-2090: A808 E8CE 0000 C168 0000 0087 6C3E 20AA .....h....l> .
03-2098: 5343 5241 5443 4820 4B45 5920 A809 6C07 SCRATCH KEY ..l.
03-20A0: 0000 444D 0761 0834 474B F082 C0CE 449C ..DM.a.4GK....D.
03-20A8: F081 F081 20B5 5343 5241 5443 4820 5020 .... SCRATCH P
03-20B0: A808 E8CE 0000 0022 6C20 20C0 5343 5241 ..... "l .SCRA
```

Принципы фон Неймана

3. Принцип однородности памяти.

Программы и 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.

Принципы фон Неймана

4. Принцип последовательного программного управления.

Все задачи, должны быть представлены в виде программы, которая состоит из набора команд. Команды представляют собой закодированные управляющие слова и выполняются процессором автоматически друг за другом в определенной последовательности.

Принцип программного управления определяет общий механизм автоматического выполнения программы.

```
section .text
_start:
    mov     rax, 1
    mov     rdi, 1
    mov     rsi, message
    mov     rdx, 13
    syscall

write

    mov     eax, 60
    xor     rdi, rdi
    syscall

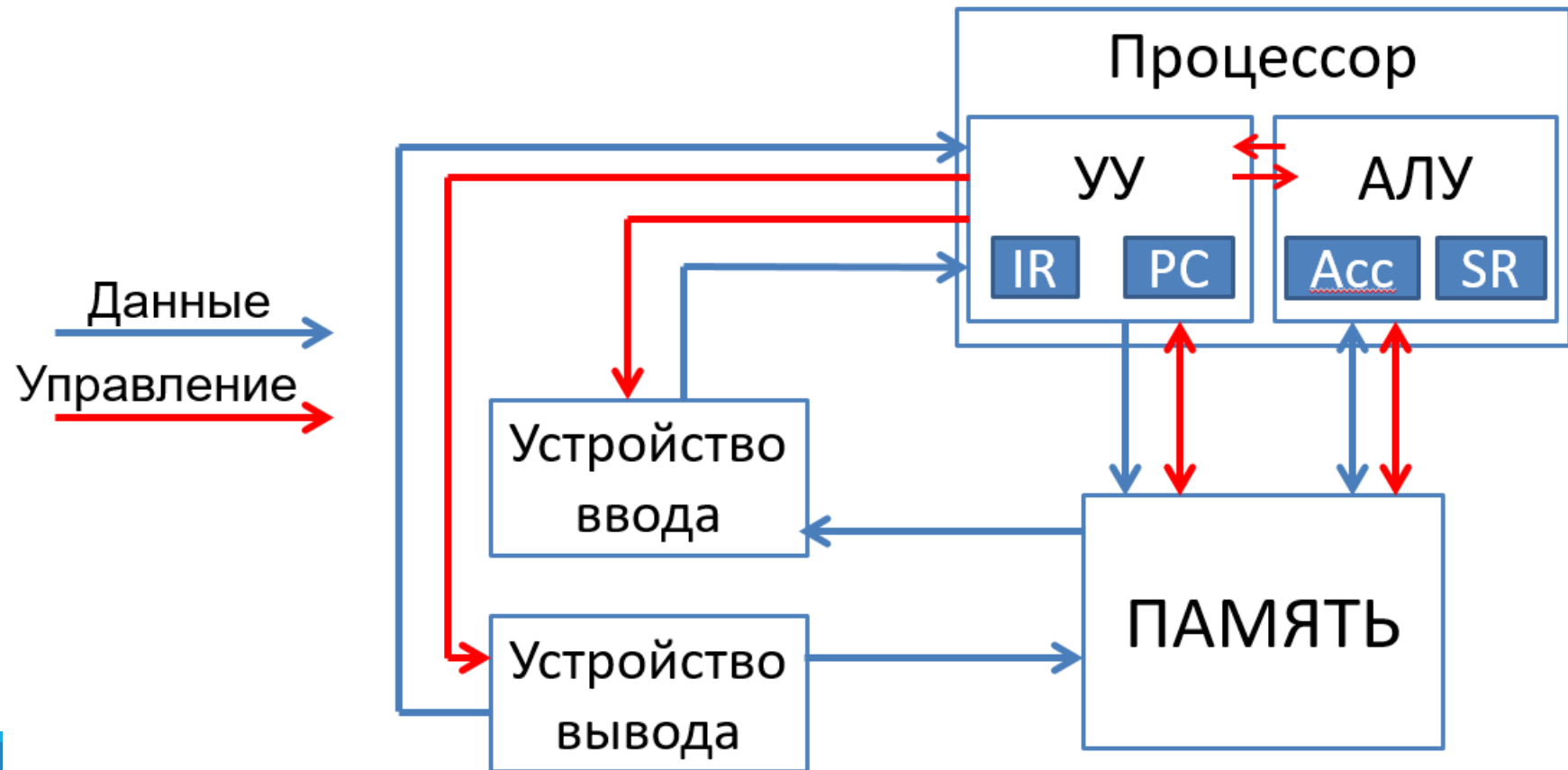
message:
    db      "Hello, World", 10
```


Принципы фон Неймана

5. Принцип жесткости архитектуры.

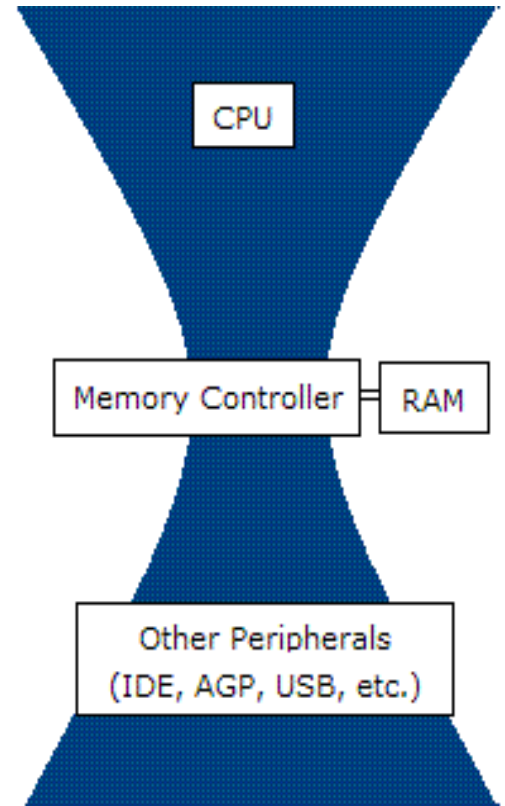
Неизменяемость в процессе работы топологии, архитектуры, списка команд ВМ.

Структура машины фон Неймана



Узкое место архитектуры фон Неймана

- Совместное использование шины для памяти программ и памяти данных приводит к узкому месту архитектуры фон Неймана.
- Память программ и память данных не могут быть доступны в одно и то же время, пропускная способность является значительно меньшей, чем скорость, с которой процессор может работать.
- Это серьезно ограничивает эффективное быстродействие.



Гарвардская архитектура

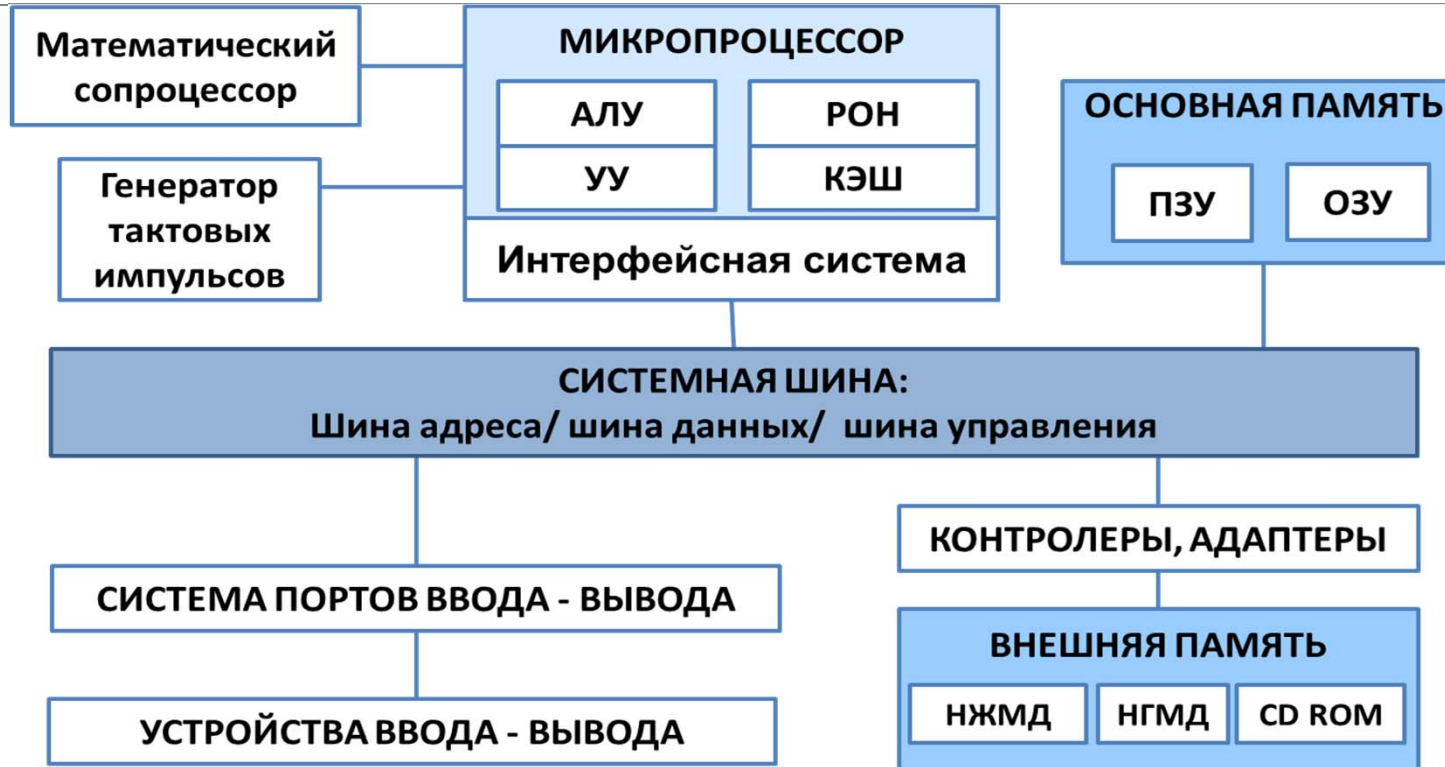
В конце 1930-х годов в Гарвардском университете Говардом Эйкенем была разработана архитектура компьютера Марк I, в дальнейшем называемая по имени этого университета.

Отличительные признаки гарвардской архитектуры:

- Память для хранения инструкций микропроцессора и память для данных представляют собой разные физические устройства.
- Канал инструкций и канал данных также физически разделены.

В гарвардской архитектуре принципиально невозможно осуществить операцию записи в память программ, что исключает возможность случайного разрушения управляющей программы в случае ошибки программы при работе с данными.

Структурная схема персонального компьютера



Совокупность функциональных элементов ПК и связи между ними

Многоуровневая компьютерная организация

Термин «**архитектура вычислительных систем**» можно интерпретировать как *распределение функций, реализуемых системой, по ее уровням и определение интерфейсов между этими уровнями.*

Архитектура вычислительной системы предполагает многоуровневую, иерархическую организацию.

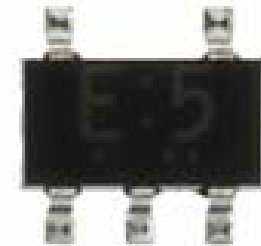
Шестиуровневое представление компьютера



Цифровой логический уровень

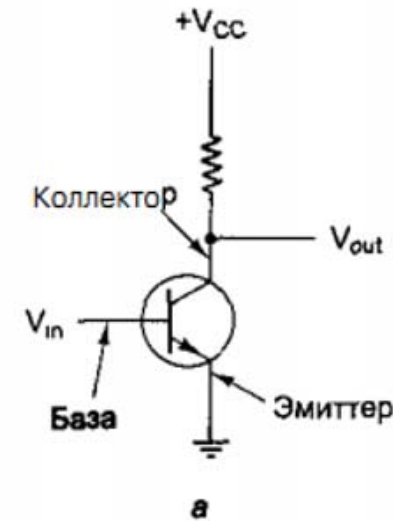
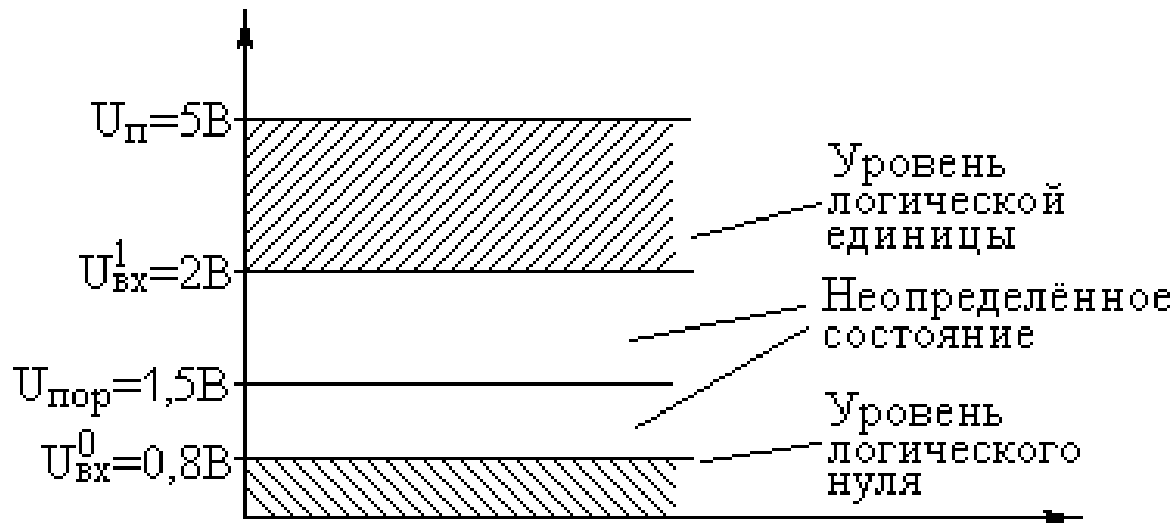
Цифровой логический уровень или аппаратное обеспечение компьютера составляют **цифровые схемы**, которые могут конструироваться из небольшого числа простых элементов путем сочетания этих элементов в различных комбинациях.

Логический вентиль — базовый элемент цифровой схемы, выполняющий элементарную логическую операцию, преобразуя таким образом множество входных логических сигналов в выходной логический сигнал.



В настоящее время в цифровых устройствах доминируют электронные логические вентили на базе **транзисторов**

Цифровой логический уровень

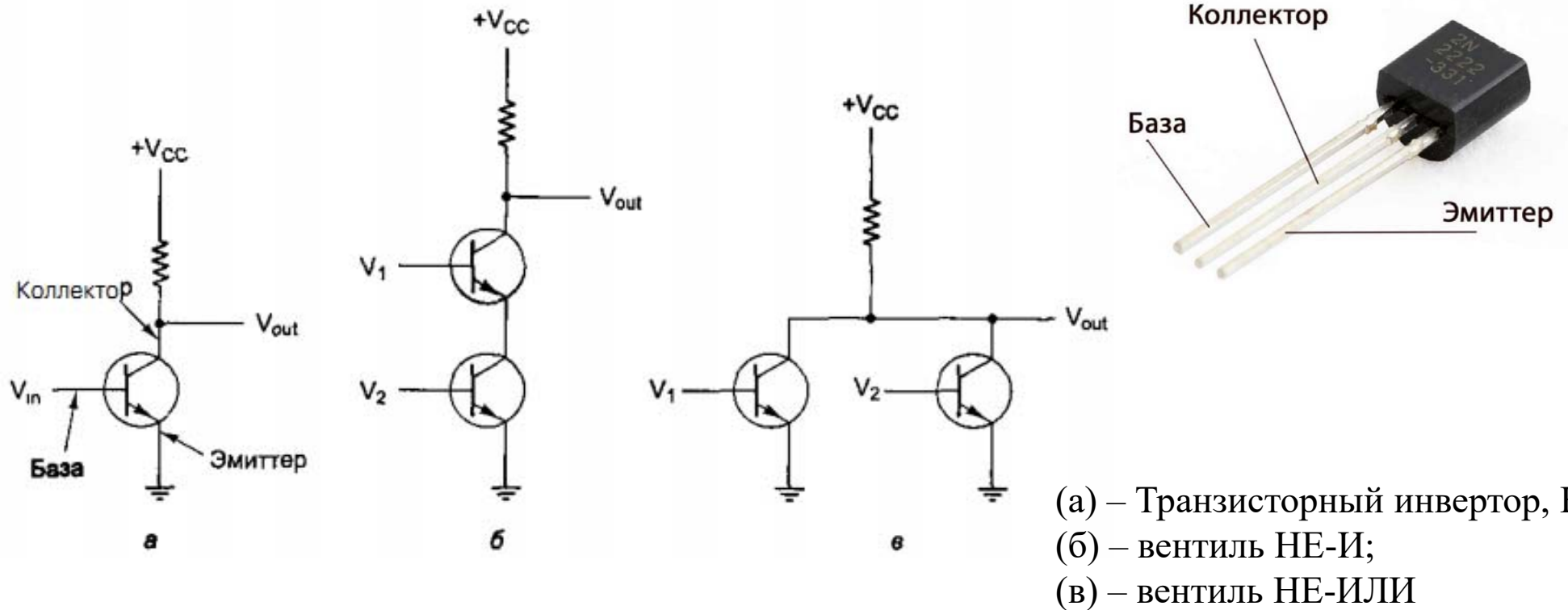


Если входное напряжение V_{in} ниже определенного критического значения, транзистор выключается и действует как сопротивление. Это приводит к напряжению на V_{out} примерно 5 В.

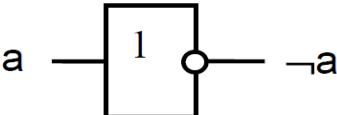
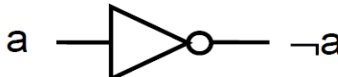
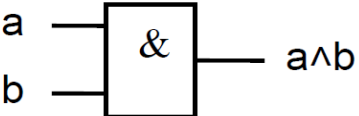

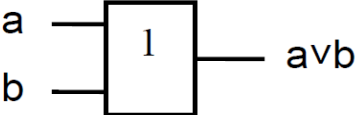
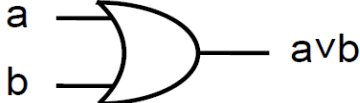
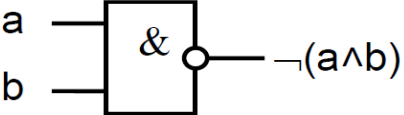
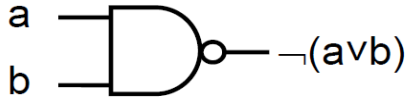
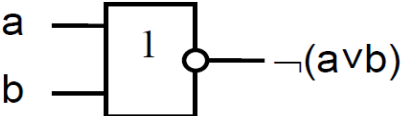
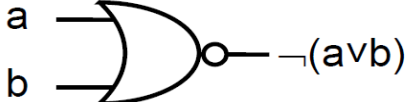
Если V_{in} большое, транзистор действует как проводник, вызывая заземление сигнала V_{out} (это 0 В).

Эта схема является **инвертором**, превращающим логический 0 в логическую 1 и наоборот.

Цифровой логический уровень



Цифровой логический уровень

Логический вентиль	ГОСТ 2.734-91	US ANSI 91-1984
Инвертор (НЕ)		
Конъюнктор (И)		
Дизъюнктор (ИЛИ)		
Элемент Шеффера (И-НЕ) (Штрих Шеффера)		
Элемент Пирса (ИЛИ-НЕ) (Стрелка Пирса)		

Условно-графические обозначения основных логических вентилей

Цифровой логический уровень

Вентиль **И**



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Вентиль **ИЛИ**



A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Вентиль **НЕ**



A	X
0	1
1	0

Вентиль **НЕ-И**



A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Вентиль **НЕ-ИЛИ**

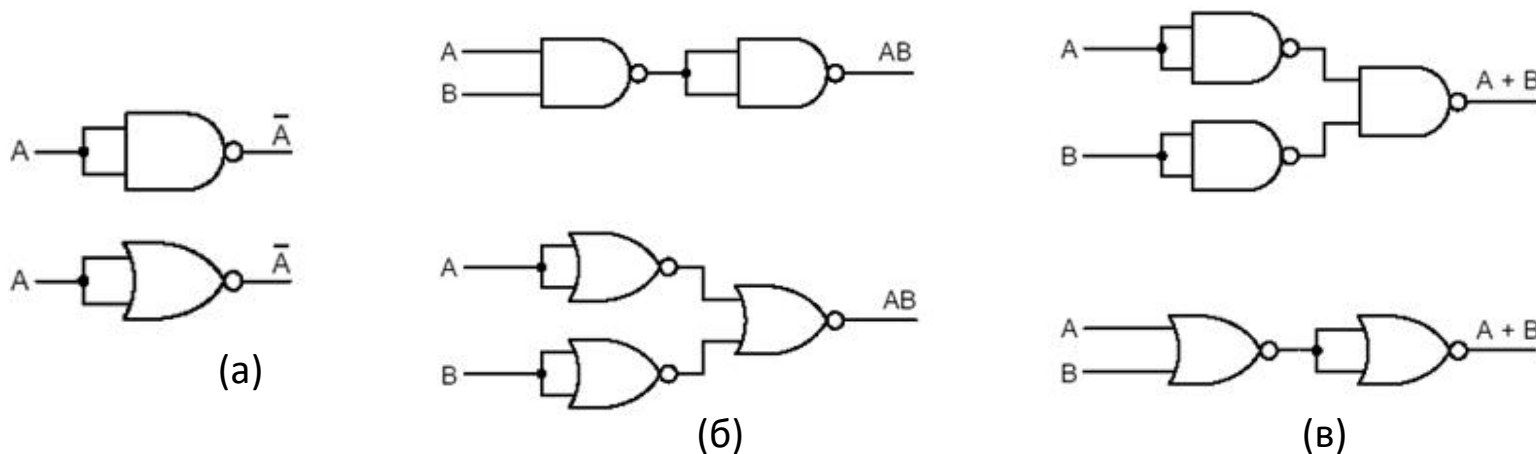


A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Таблицы истинности

Цифровой логический уровень

Для построения вентилях **НЕ-И** и **НЕ-ИЛИ** требуются по два транзистора, а для вентилях **И** и **ИЛИ** — по три транзистора на каждый вентиль. По этой причине во многих компьютерах используются вентиля **НЕ-И** и **НЕ-ИЛИ**, а не **И** и **ИЛИ**.

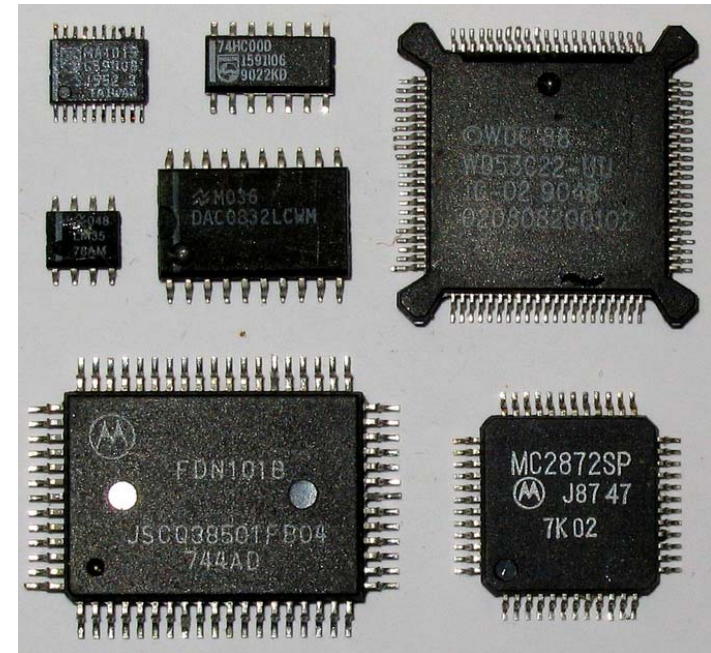


Конструирование вентилях **НЕ** (а), **И** (б) и **ИЛИ** (в) с использованием только вентилях **НЕ-И** или только вентилях **НЕ-ИЛИ**

Цифровой логический уровень

Вентили производятся и продаются не по отдельности, а в модулях, которые называются **интегральными схемами** (ИС) или микросхемами. Интегральная схема представляет собой квадратный кусочек кремния, на котором находится несколько вентилях.

- МИС (малая интегральная схема): от 1 до 10 вентиляей
- СИС (средняя интегральная схема): от 1 до 100 вентиляей
- БИС (большая интегральная схема): от 100 до 100 000 вентиляей
- СБИС (сверхбольшая интегральная схема): более 100 000 вентиляей



Уровень микроархитектуры

Задача – интерпретация команд следующего уровня архитектуры команд.

На этом уровне рассматривается взаимодействие логических схем, например арифметико-логического устройства (АЛУ), оперативной памяти, регистров.

Регистры вместе с АЛУ формируют тракт данных, обеспечивающий тот или иной алгоритм выполнения арифметической или логической операции.

Работа тракта данных регулируется микропрограммой или аппаратным обеспечением.

Уровень микроархитектуры

Процесс обработки данных в классическом упрощённом виде, состоит из нескольких этапов:

- 1. выборка команды;**
- 2. формирование адреса следующей команды;**
- 3. декодирование команды;**
4. вычисление адресов операндов;
5. выборку операндов;
- 6. исполнение операции;**
7. запись результата;
8. Переход к этапу 1.

Уровень микроархитектуры

Пример. Допустим, **ADD** – команда сложения двух значений, хранящихся в памяти в ячейках с адресами **1012** и **4501** и запись результата в ячейку **1012**.

ADD	1012	4501
-----	------	------

Первый такт: считывание кода операции в специальный регистр команд АЛУ;

Второй такт: считывание из ячейки 1012 ОЗУ первого слагаемого и перемещение его в АЛУ;

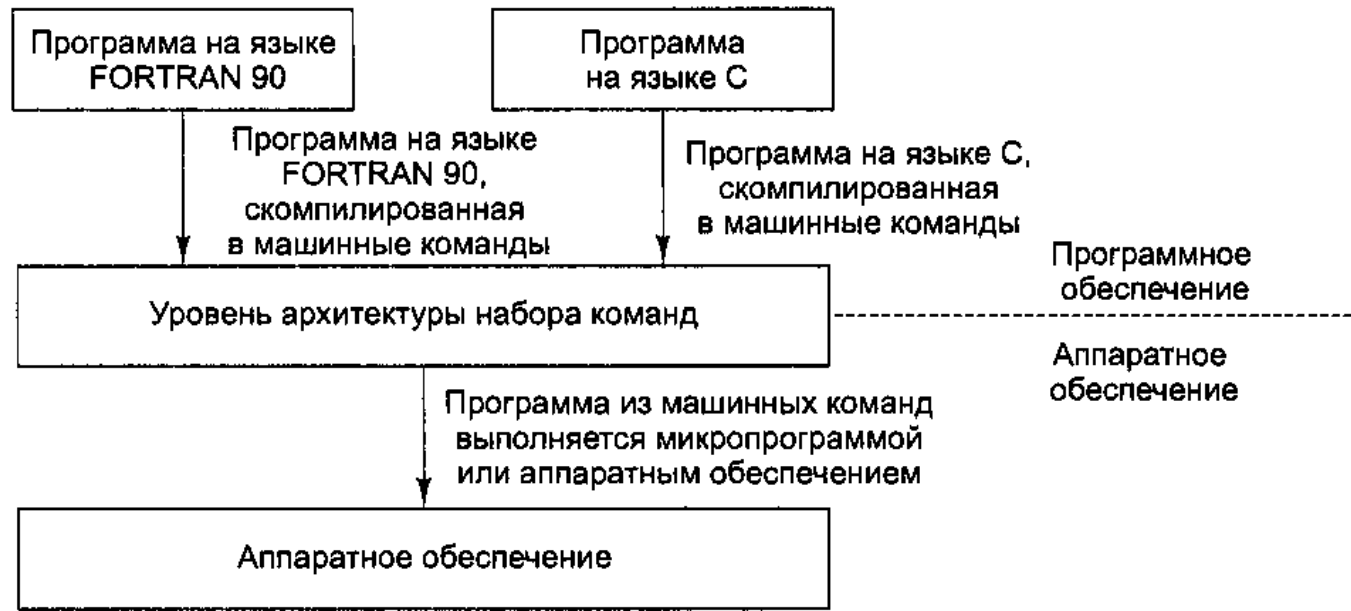
Третий такт: считывание из ячейки 4501 ОЗУ второго слагаемого и перемещение его в АЛУ;

Четвертый такт: сложение в АЛУ переданных туда чисел и формирование суммы;

Пятый такт: считывание из АЛУ суммы чисел и запись ее в ячейку 1012 ОЗУ.

Уровень архитектуры системы команд

Задача — связывает аппаратные средства компьютера с его программным обеспечением.



Уровень архитектуры системы команд

Система команд (также набор команд) — соглашение о предоставляемых архитектурой ЭВМ средствах программирования, а именно: определённых типах данных, инструкций, системы регистров, методов адресации, моделей памяти, способов обработки прерываний и исключений, методов ввода и вывода.

Все машинные команды можно разделить на группы по видам выполняемых операций:

- операции пересылки информации внутри ЭВМ;
- арифметические операции над информацией;
- логические операции над информацией;
- операции обращения к внешним устройствам ЭВМ;
- операции передачи управления;
- обслуживающие и вспомогательные операции.

Уровень архитектуры системы команд

Формат команды

Машинная команда состоит из двух частей: **операционной** и **адресной**.



Операционная часть команды – это группа разрядов в команде, предназначенная для предоставления кода операции машины.

Адресная часть команды – это группа разрядов, в которых записываются коды адреса ячеек памяти машины. Часто эти адреса называются адресами операндов, т.е. чисел, участвующих в операции.

Уровень архитектуры системы команд

По количеству адресов, записываемых в команде, команды делятся на безадресные, одно -, двух- и трехадресные.

Типовая структура трехадресной команды:

КОП	Адрес 1-го операнда	Адрес 2-го операнда	Адрес результата
------------	----------------------------	----------------------------	-------------------------

Типовая структура двухадресной команды:

КОП	Адрес 1-го операнда	Адрес 2-го операнда
------------	----------------------------	----------------------------

Типовая структура одноадресной команды:

КОП	Адрес операнда
------------	-----------------------

Типовая структура безадресной команды:

КОП	Расширение кода операции
------------	---------------------------------

Уровень архитектуры системы команд

AT&T-ассемблер

```
subl 0x20(%ebx,%ecx,0x4),%eax  
movl $0x10,%ebx
```

Intel-ассемблер

```
sub eax,[ebx+ecx*4h-20h]  
mov ebx,10h
```

Уровень архитектуры системы команд

Современные ЭВМ выполняют несколько сотен различных команд, структура команд, их сложность и длина определяют архитектуру процессора.

- Архитектура с полным набором команд **CISC** (Complex Instruction Set Computer)
- Архитектура с ограниченным набором – **RISC** (Reduced Instruction Set Computer).

CISC архитектура

CISC — концепция проектирования процессоров, которая характеризуется следующим набором свойств:

- наличие в процессоре сравнительно небольшого числа регистров общего назначения;
- большое количество машинных команд;
- разнообразие способов адресации операндов;
- множество форматов команд различной разрядности;
- наличие команд, где обработка совмещается с обращением к памяти.

Типичными представителями являются процессоры компании Intel серии 8086 и Pentium и процессоры Motorola MC680x0.

RISC архитектура

RISC — архитектура процессора, в котором быстродействие увеличивается за счёт упрощения инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим.

- Фиксированная длина машинных.
- Сокращение числа форматов команд и их простота.
- Использование ограниченного количества способов адресации.
- Специализированные команды для операций с памятью — чтения или записи.
- Большое количество регистров общего назначения.

Типичными представителями являются процессоры Alpha фирмы DEC, серии PA фирмы Hewlett-Packard, семействе PowerPC и т. п.

Уровень операционной системы

Операционная система — это программа, которая добавляет ряд команд и особенностей к тем, которые обеспечиваются уровнем команд.

Набор команд уровня операционной системы — это полный набор команд, доступных для прикладных программистов: все команды более низкого уровня, а также команды, которые называются системными вызовами.

Особенности уровня:

- виртуальная память
- параллельная обработка
- файл ввода-вывода

Уровень языка ассемблера

Уровень языка ассемблера отличается от трех предыдущих тем, что он реализуется с помощью трансляции, а не с помощью интерпретации.

Причины использования языка ассемблера:

1. программист может составить гораздо меньшую по размеру программу, которая будет работать гораздо быстрее, чем программа, написанная на языке высокого уровня.
2. некоторым процедурам требуется полный доступ к аппаратному обеспечению, что обычно невозможно сделать на языке высокого уровня.

Структура команды в языке ассемблера отражает структуру соответствующей машинной команды и языки ассемблера для разных машин могут отличаться.

Язык ассемблера

Язык ассемблера (*assembly language*) – машинно-ориентированный язык низкого уровня. Т.е. он больше любых других приближен к архитектуре ЭВМ и ее аппаратным возможностям, что позволяет получить к ним более полный доступ, нежели в языках высокого уровня, таких как C/C++, Perl, Python и пр.

Таким образом язык ассемблера – это язык, с помощью которого понятным для человека образом пишутся команды процессора. Стоит отметить, что процессор понимает не команды ассемблера, а последовательности из нулей и единиц. Преобразование команд с языка ассемблера в исполняемый машинный код осуществляется специальной программой транслятором – **Ассемблер**.

Программы, написанные на языке ассемблера не уступают в качестве и скорости программам, написанным на машинном языке, так как транслятор просто переводит мнемонические обозначения команд в последовательности байтов (нулей и единиц).

Язык ассемблера

Для каждого процессора существует свой ассемблер и соответственно, свой язык ассемблера.

Наиболее распространенными ассемблерами для архитектуры x86 являются:

Для **DOS/Windows**: Borland Turbo Assembler (**TASM**), Microsoft Macro Assembler (**MASM**) и Watcom Assembler (**WASM**).

Для **GNU/Linux**: **gas** (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис.

NASM (Netwide Assembler) – это открытый проект ассемблера, версии которого доступны под различные операционные системы, и который позволяет получать объектные файлы для этих систем. В NASM используется Intel-синтаксис и поддерживаются инструкции x86-64.

Формат команд NASM

Каждая машинная команда состоит из двух частей:

- операционной - определяющей, "что делать";
- операндной - определяющей объекты обработки, "с чем делать".

Типичный формат записи команд NASM имеет вид:

[метка:] мнемокод [операнд {,операнд}] [;комментарий]

Формат команд NASM

Мнемокоды

Мнемокод — непосредственно мнемоника инструкции процессору и является обязательной частью команды. Например, пересылка (`mov`) или сложение (`add`).

Операнды

Операнд— объект, над которым выполняется машинная команда или оператор языка программирования.

Команда может иметь один или два операнда, или вообще не иметь операндов. Число операндов неявно задается кодом команды.

Примеры:

Нет операндов	<code>ret</code>	; Вернуться (используется в подпрограммах)
Один операнд	<code>inc ecx</code>	; Увеличить на 1 ecx
Два операнда	<code>add eax, 12</code>	; Прибавить 12 к eax

Формат команд NASM

Метки

Метка – идентификатор, значением которого является адрес первого байта того предложения исходного текста программы, которое он обозначает. Метка в языке ассемблера может содержать следующие символы:

- все буквы латинского алфавита;
- цифры от 0 до 9;
- спецсимволы: `_`, `@`, `$`, `?`.

Все метки, которые записываются в строке, не содержащей директиву ассемблера, должны заканчиваться двоеточием `:`.

Комментарии

Комментарии отделяются от исполняемой строки символом `;`. При этом все, что записано после символа точка с запятой и до конца строки, является комментарием.

Команды mov

Команда MOV копирует данные из операнда-источника в операнд-получатель.

MOV <получатель>, <источник>

Варианты использования команды MOV с разными операндами

MOV reg, reg

MOV mem, imm

MOV mem, reg

MOV reg, imm

MOV reg, mem

~~MOV mem, mem~~ ❌

Примеры:	mov eax, 403045h	; пишет в eax значение 403045
	mov cx, [eax]	; помещает в регистр CX значение (размера word) из памяти указанной в EAX (403045)
	mov eax, ebx	; Пересылаем значение регистра EBX в регистр EAX
	mov x, 0	; Записываем в переменную x значение 0
	mov al, 1000h	; Ошибка – попытка записать 2-байтное число в 1-байтный регистр
	mov eax, cx	; Ошибка – размеры операндов не совпадают

Команда `int 0x80`

`int 0x80` — генерирует программное прерывание с номером 0x80 для системного вызова ядра. Номер системного вызова передается ядру через регистр **EAX**, параметры системных вызовов задаются в регистрах **EBX**, **ECX**, **EDX**, **ESI** и **EDI**

Системный вызов 'write'	<code>mov eax,4</code>
Параметры	
- номер потока вывода	например, вывод на экран <code>mov ebx,1</code>
- адрес памяти с данными для вывода	<code>mov ecx,msg</code>
- количество данных в байтах	<code>mov edx,len</code>
Системный вызов 'exit'	<code>mov eax,1</code>
Параметр	
- код завершения	<code>mov ebx,0</code>
Системный вызов 'read'	<code>mov eax,3</code>
Параметры	
- номер потока ввода	например, ввод с клавиатуры <code>mov ebx,0</code>
- адрес памяти, в которой надо разместить прочитанные данные	<code>mov ecx,string</code>
- количество данных в байтах	<code>mov edx,4</code>

Объявление инициализированных данных

Директивы	Назначение	пример	
Резервируют память и указывают, какие значения должны храниться в этой памяти			
DB	Для определения 1-байтовых значений (8 бит)	<code>msg db "Hello"</code>	По адресу msg располагается строка из 5 символов (т.е. массив 1-байтовых ячеек, содержащих коды символов)
DW	Для определения 2-байтовых значений – слов (16 бит)	<code>count dw 7</code>	По адресу count располагается 2-байтовое слово, в которое занесено число 7
DD	Для определения двойных слов (32 бита)		

Объявление неинициализированных данных

Директивы	Назначение	пример	
Сообщают ассемблеру, что необходимо зарезервировать заданное количество ячеек памяти			
RESB	Резервирование заданного числа 1-байтовых ячеек	<code>string resb 20</code>	По адресу с меткой <code>string</code> , будет расположен массив из 20 1-байтовых ячеек (хранение строки символов)
RESW	Резервирование заданного числа 2-байтовых ячеек (слов)	<code>count resw 256</code>	По адресу с меткой <code>count</code> , будет расположен массив из 256 2-байтовых слов
RESD	Резервирование заданного числа 4-байтовых ячеек (двойных слов)	<code>x resd 1</code>	По адресу с меткой <code>x</code> будет расположено одно двойное слово (т.е. 4 байта для хранения большого числа)

Структура программы NASM

SECTION .data	; секция содержит переменные для
...	; которых задано начальное значение
SECTION .bss	; секция содержит переменные для
...	; которых не задано начальное значение
SECTION .text	; секция содержит код программы
global _start	
_start:	; Точка входа в программу
...	
mov eax,1	; системный вызов '_exit'
mov ebx,0	; Код завершения 0 (success)
int 0x80	; Вызов ядра

"Hello, world!"

```
SECTION .data
    msg     db "Hello, world!",0xa           ; Строка
    len     equ $ - msg                     ; Размер строки
SECTION .text
global _start
_start:                                       ; Точка входа в программу
    mov     eax, 4                          ; системный вызов 'write'
    mov     ebx, 1                          ; Дескриптор 1 (stdout)
    mov     ecx, msg                       ; Указатель на данные
    mov     edx, len                       ; Размер данных
    int     0x80                           ; Вызов ядра
    mov     eax, 1                          ; '_exit' системный вызов
    mov     ebx, 0                          ; код завершения
    int     0x80                           ; Вызов ядра
```

Компиляция и линковка

`nasm -f elf64 hello.asm` # Создание объектного файла

`ld -o hello hello.o` # Создание исполняемого файла

`./hello` # запуск программы

Memory dump

```
> dump $2000 $2200
03-2000: 4083 130D 1311 1314 021F 2023 4C49 5354 0..... #LIST
03-2008: 204B 4559 A807 6C0F 0000 0828 47D1 443A KEY...1....<G.D:
03-2010: 44E0 0360 F187 607C 2034 321D 07EC 0A1D D...`...`! 42.....
03-2018: 47D7 F082 44F6 6C05 6D2D 4526 E8C6 F081 G...D..l..m-E&....
03-2020: 4523 F081 F081 2060 4C49 5354 A805 6C2C E#.... `LIST...l,
03-2028: 0000 0A34 C12C 082A 47B3 441C 7171 834B ...4...*G.D.qq.K
03-2030: 2025 B800 7171 834B 2026 321D 0001 B21D %...qq.K &2.....
03-2038: 834B 2025 8000 F020 2001 F403 47B8 6FBC .K %... ..G.o.
03-2040: 0360 F187 607C 2038 321D 07BB 0A1D 47A8 `...`! 82.....G.
03-2048: F082 7170 3360 7171 834B 2023 B800 7171 ..qp3`qq.K #...qq
03-2050: 2021 B800 F081 44BD 6C09 6D03 C103 E8C6 ?.....D..l..m.....
03-2058: 4504 6D01 C103 E8C6 F081 C103 6CFA 6FF9 E.m.....l..o.
03-2060: 2077 5255 4E20 A805 6C0F 0000 4406 F081 wRUN ..l...D...
03-2068: 0796 086C 4785 F082 0824 4772 7171 834B ...lG....$Grqq.K
03-2070: 2023 B800 F082 C0CE C103 6CE3 F081 208A #.....l...
03-2078: 434F 4E54 A805 6C08 0000 47EF F081 077E CONT..l...G....~
03-2080: 086C 476E F082 C0CE C103 6C02 F081 0887 .lGn.....l.....
03-2088: C0D3 F081 2097 5343 5241 5443 4820 4120 .... .SCRATCH A
03-2090: A808 E8CE 0000 C168 0000 0087 6C3E 20AA .....h....l> .
03-2098: 5343 5241 5443 4820 4B45 5920 A809 6C07 SCRATCH KEY ..l.
03-20A0: 0000 444D 0761 0834 474B F082 C0CE 449C ..DM..a..4GK....D.
03-20A8: F081 F081 20B5 5343 5241 5443 4820 5020 .... .SCRATCH P
03-20B0: A808 E8CE 0000 0022 6C20 20C0 5343 5241 ..... "l .SCRA
```

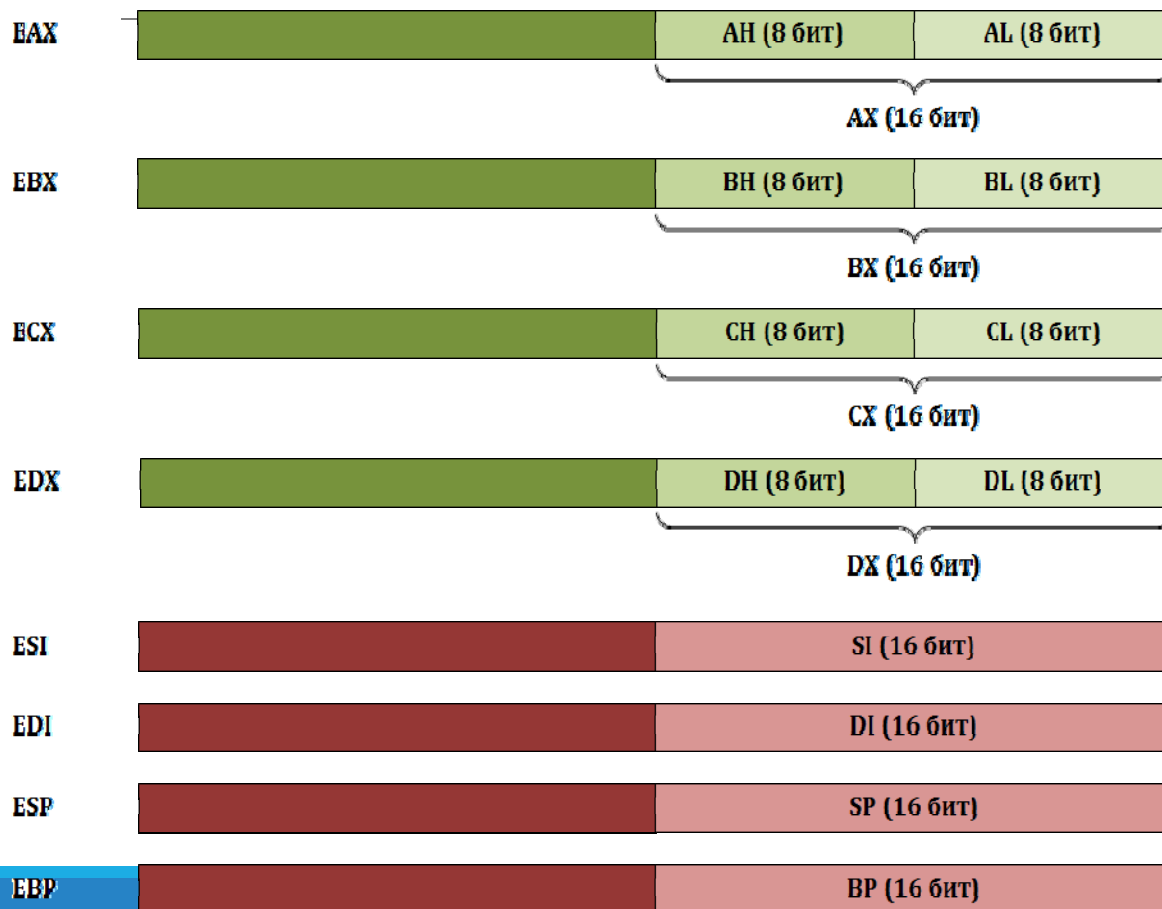

Регистры

Регистры – это специальные ячейки памяти, расположенные непосредственно в процессоре. Работа с регистрами выполняется намного быстрее, чем с ячейками оперативной памяти.

Регистры можно разделить на

- регистры общего назначения,
- указатель команд,
- регистр флагов,
- сегментные регистры.

Регистры общего назначения



Регистры ESI, EDI, ESP и EBP позволяют обращаться к младшим 16 битам по именам SI, DI, SP и BP соответственно, а регистры EAX, EBX, ECX и EDX позволяют обращаться как к младшим 16 битам (по именам AX, BX, CX и DX), так и к двум младшим байтам по отдельности (по именам AH/AL, BH/BL, CH/CL и DH/DL).

Регистры общего назначения

EAX/AX/AH/AL (*accumulator register*) – аккумулятор;

EBX/BX/BH/BL (*base register*) – регистр базы;

ECX/CX/CH/CL (*counter register*) – счётчик;

EDX/DX/DH/DL (*data register*) – регистр данных;

ESI/SI (*source index register*) – индекс источника;

EDI/DI (*destination index register*) – индекс приёмника (получателя);

ESP/SP (*stack pointer register*) – регистр указателя стека;

EBP/BP (*base pointer register*) – регистр указателя базы кадра стека.

Несмотря на существующую специализацию, все регистры можно использовать в любых машинных операциях. Однако надо учитывать тот факт, что некоторые команды работают только с определёнными регистрами. Например, команды умножения и деления используют регистры EAX и EDX для хранения исходных данных и результата операции. Команды управления циклом используют регистр ECX в качестве счётчика цикла.

Указатель команд

Регистр EIP (*указатель команд*) содержит смещение следующей подлежащей выполнению команды. Этот регистр непосредственно недоступен программисту, но загрузка и изменение его значения производятся различными командами управления, к которым относятся команды условных и безусловных переходов, вызова процедур и возврата из процедур.

Уровень языка ассемблера

**; Hello World for Intel
; Assembler (Linux)**

```
SECTION .data
msg db "Hello, world!",0xa
len      equ $ - msg
SECTION .text
global _start
_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, msg
    mov edx, len
    int 0x80
    mov eax, 1
    mov ebx, 0
    int 0x80
```

**; Hello World for Intel
; Assembler (MSDOS)**

```
mov ax,cs
mov ds,ax
mov ah,9
mov dx, offset Hello
int 21h
xor ax,ax
int 21h

Hello:
    db "Hello
World!",13,10,"$"
```

<http://helloworldcollection.de/>

**; Hello World in Assembler for
; the DEC PDP-11 with the
;RSX-11M-PLUS operating system**

```
;
                                .title Hello
                                .ident /V0001A/
                                .mcall qiow$, exit$$
                                .psect $code,ro,i
start:  qiow$ #5,#5,,, <#str,
#len, #40>
                                exit$$
                                .psect $data,ro,d
str:    .ascii / Hello World!/
len=.-str
                                .end start
```

Уровень языка высокого уровня

Язык программирования высокого уровня — язык программирования, разработанный для быстроты и удобства использования программистом. Основная черта языков программирования высокого уровня — это абстракция, то есть введение смысловых конструкций, кратко описывающих такие структуры данных и операции над ними, описания которых на машинном коде (или другом низкоуровневом языке программирования) очень длинны и сложны для понимания.

Примеры высокоуровневых языков программирования: C, C++, C#, Visual Basic, Java, Python, PHP, Ruby, Perl, Delphi (Pascal).

Выводы

- Классическая структура ЭВМ отвечает модели Фон Неймана. Современные ЭВМ далеко ушли от этой модели, но по-прежнему имеют большое число общих черт, например – двоичная система счисления, управление потоком команд.
- Решение задач на ЭВМ реализуется программным способом, т.е. путем выполнения последовательно во времени отдельных операций над информацией, предусмотренных алгоритмом решения задачи.
- Архитектура вычислительной системы предполагаем многоуровневую, иерархическую организацию.
- Компьютер проектируется как иерархическая структура уровней, которые надстраиваются друг над другом. Каждый уровень представляет собой определенную абстракцию различных объектов и операций.