Организация ЭВМ и Ассемблер

Александра Игоревна Кононова / **illinc@mail.ru**
+7-985-148-32-64 (телефон), +7-977-977-97-29 (WhatsApp),
gitlab.com/illinc/raspisanie

МИЭТ

5 февраля 2022 г. — актуальную версию можно найти на
https://gitlab.com/illinc/gnu-asm

Цель курса

Формирование подкомпетенции ПК-5.ОЭА Способен программировать на ассемблере при решении практических задач

Ассемблер изучается на примере системы команд IBM PC (ПК: x86 и x86-64), диалекта AT&T (GNU Assembler).

Требуется:

1. распознавать 0 в любом формате, а 1 и (-1) — в целочисленном любого размера;
2. знать основные инструкции IBM PC и где искать остальные;
3. уметь программировать на ассемблере IBM PC (AT&T, app, 32/64);
4. уметь совмещать ассемблер и C++ (GCC);
5. отличать MS Windows от GNU/Linux|BSD|MacOS X, 32 от 64, MS VS от GCC, диалекты Intel от AT&T, диалект MS VS от C++.

Оценивание (подробно на gitlab)

Регламент полностью — в начале приложения А `https://gitlab.com/illinc/gnu-asm/-/raw/master/gnu-asm-theory-labs.pdf?inline=false`

Делать л/р дома можно и нужно! Оценка автоматом по баллам: $86/70/50$

1. выполнение л/р или курсового проекта;
2. решение задач на лекциях: $2-4$ балла за задачу;
3. вычитка материала: $1-8$ балла за замечание/исправление/вопрос, послужившие улучшению курса.

Штрафы к л/р — опоздание $\geqslant 2$ занятия, -1 за каждое; качество.

GNU Compiler Collection — коллекция GCC (под MS Windows — MinGW)

*nix — QT Creator и др. + GCC; либо консоль + GCC + GDB

MS Windows — QT Creator + MinGW (бесплатные, свободные, качать комплексный дистрибутив среда+компилятор); или другая свободная IDE с MinGW.
Использование MS Visual Studio недопустимо: она не поддерживает MinGW.

GCC онлайн (GNU/Linux, x86-64):
`https://godbolt.org/` Output → Run the compiled output
`https://www.onlinegdb.com/online_c++_compiler`

Структурная и иерархическая декомпозиции ВС

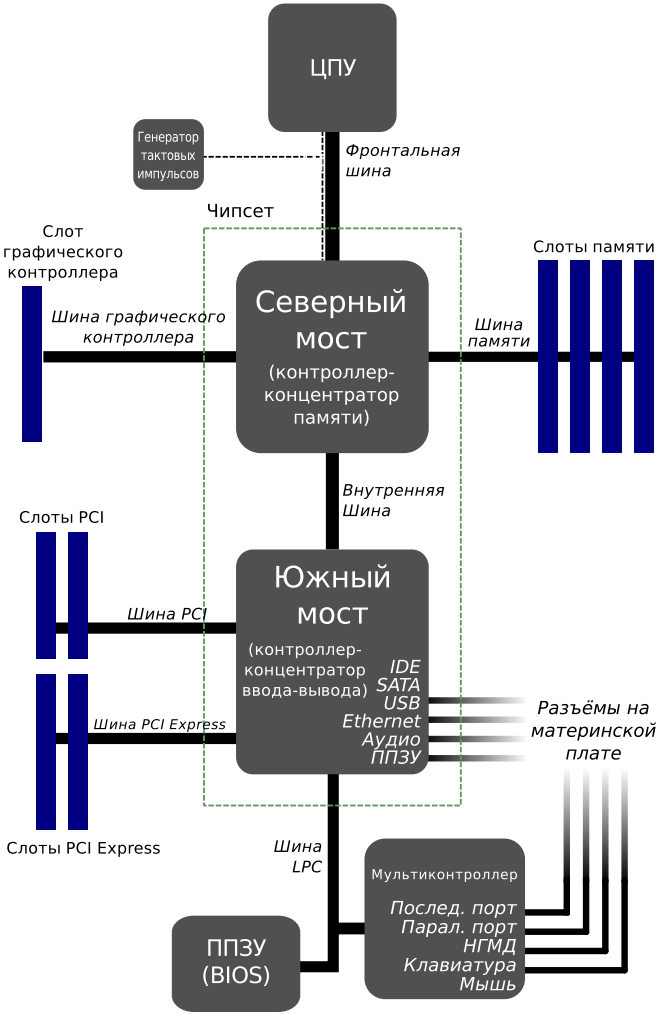
- **Архитектура системы** — фундаментальная организация системы, реализованная в её компонентах, их взаимоотношениях друг с другом и средой, а также в принципах, определяющих её конструкцию (проектирование, дизайн) и развитие (ANSI/IEEE Std 1471-2000).
- **Вычислительная система**
  1. технические средства (вычислительная машина — компьютер);
  2. человек (оператор или программист);
  3. связывающие их программные средства.
- Архитектура вычислительной системы
  1. **структурная декомпозиция** — аппаратные составляющие, связи между ними;
  2. **иерархическая декомпозиция** — логическо-информационная структура, языки взаимодействия с программными и техническими средствами системы.

Структурная декомпозиция ВС

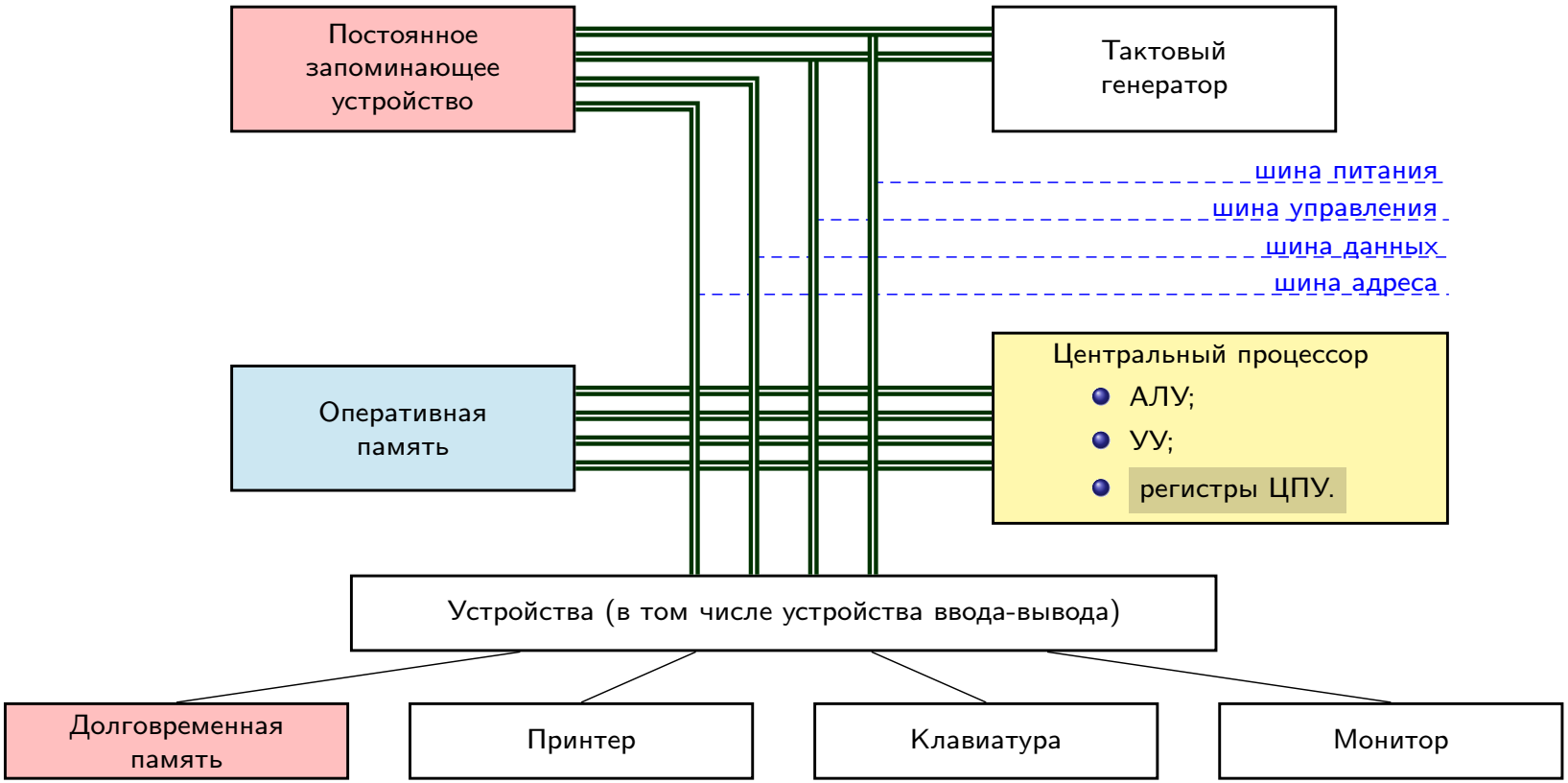
Из каких частей состоит компьютер (ПК)?

Состав персонального компьютера

1. Системный блок — внутренние устройства:
   - **системная (материнская) плата**:
     северный мост — обмен между ЦПУ и скоростными устройствами, частота шины, вид и объём ОЗУ;
     южный мост — часы, память CMOS, работа с низкоскоростными интерфейсами: SATA, USB и т. п.;
   - центральный процессор (ЦПУ);
   - оперативная память (ОЗУ);
   - долговременная (внешняя) память — ЖД, SSD;
   - видеокарта, звуковая карта и т. д.
2. Внешние устройства (в том числе ввода-вывода).



Структура системной шины



Иерархическая декомпозиция ВС (Таненбаум)

Вычислительная система как иерархия виртуальных машин M_i с языками Я$_i$ (уровней)

Над иерархией — программист $\rightarrow$ Я$_n$.

Я$_i \rightarrow$ Я$_{i-1}$:

- **компиляция** — перевод на Я$_{i-1}$ целиком;
- **интерпретация** — выполнение покомандно.

Я$_0$ интерпретируется электронными схемами (M_0).

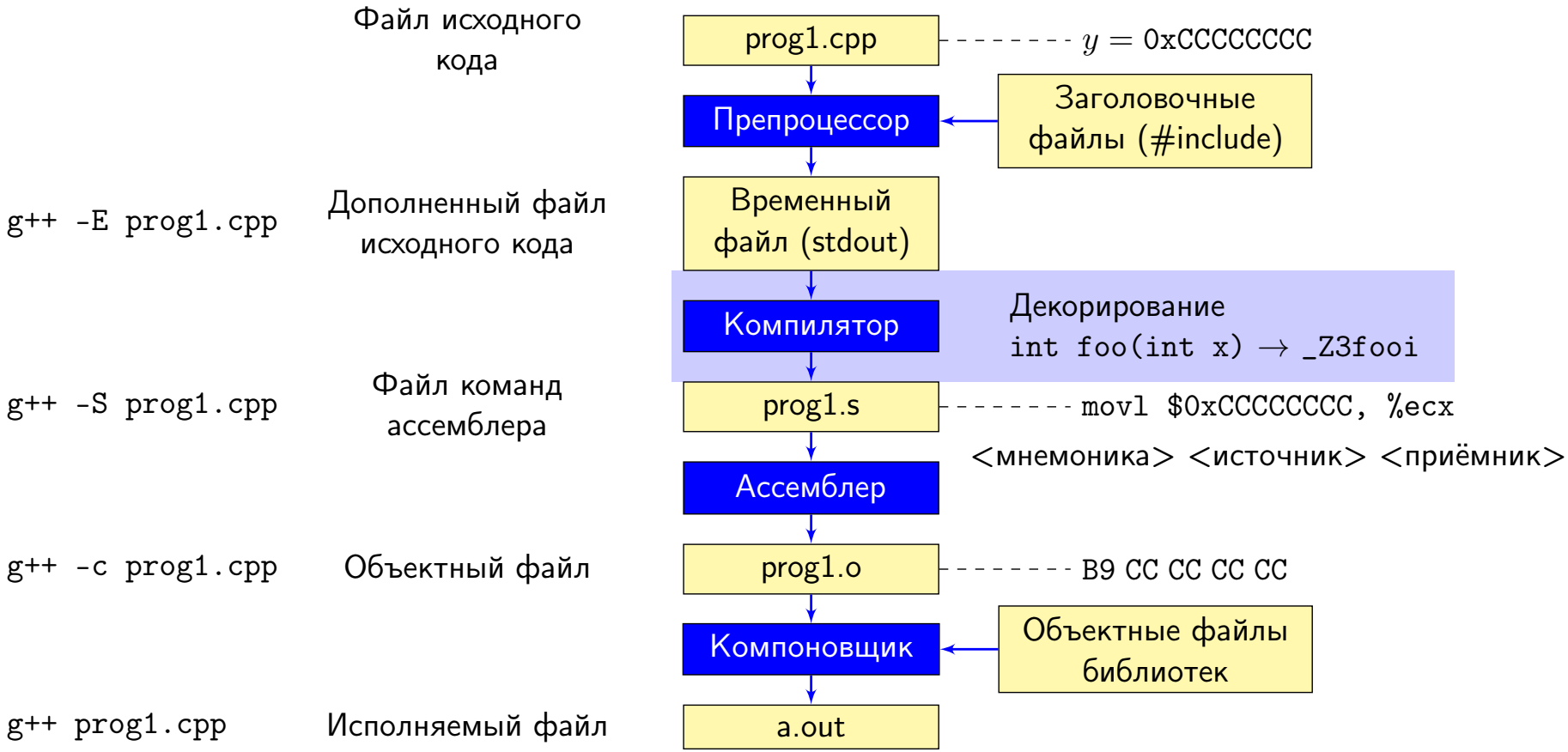
Платформа — совокупность нижележащих уровней; для программы на языке Ассемблера это компилятор Ассемблера (ассемблер), операционная система, архитектура команд и её аппаратная реализация.

Уровни ($Я_n = C{+}{+}$)

0. **Цифровой логический** уровень, античность/XVII в. — сигналы ($0/1$ — 1938) и вентили, бит (для недвоичных — трит и т. д.).
1. **Микроархитектура,** 1957 — регистры (реализация), микрокоманды, УУ, АЛУ, ЗУ, шины, слово.
2. **Архитектура команд,** XIX в./1938 — регистры (интерфейс), команды:
   - CISC *(complex instruction set computer)* — x86;
   - RISC *(reduced instruction set computer, load/store)* $\rightarrow$ *VLIW (комп-я).*

   архитектура памяти, адресация, простые форматы данных (целочисленные, с плавающей запятой), работа с устройствами, байт.
3. **Операционная система,** 1946/1956 — загрузка программ в оперативную память и их выполнение, системные вызовы, управление памятью и устройствами.
4. **Язык ассемблера,** 1949 — мнемоники (символическое представление команд), символическое представление адресации, директивы.
5. **Язык высокого уровня,** 1945/1954 — алгоритмические конструкции, переменные, сложные типы данных, **типизация.**

Сборка программы (GCC): ЯВУ → ОС



`g++ -o prog prog1.cpp` — смена названия исполняемого файла с `a.out` на `prog`

♢ Результат компиляции программы на C++

```
#include <iostream>
int foo(int x){    return 3*x + 1;  }
int main(){
    int x = 13;
    std::cout << "foo(" << x << ") = " << foo(x) << std::endl;
    return 0;
}
```

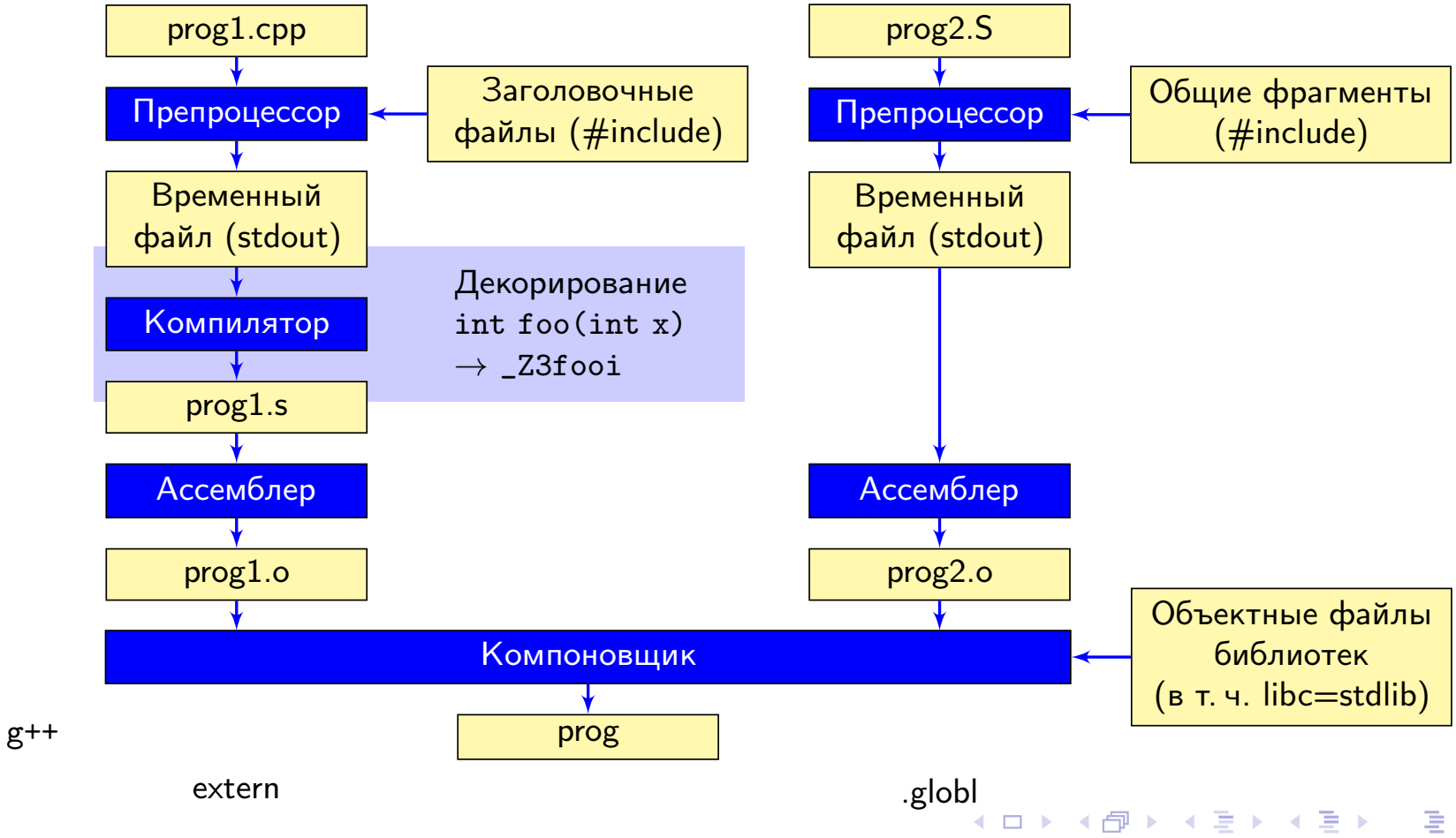
g++ <имя файла>.cpp -S останов сборки после компиляции — вместо исполняемого a.out получаем файл на языке Ассемблера <имя файла>.s:

```
_Z3fooi:
...
movl    %edi, -4(%rbp)
movl    -4(%rbp), %edx
movl    %edx, %eax
addl    %eax, %eax
addl    %edx, %eax
addl    $1, %eax
...
ret
```

```
main:
...
movl    -20(%rbp), %eax
movl    %eax, %edi
call    _Z3fooi
movl    %eax, %esi
movq    %rbx, %rdi
call    _ZNSolsEi@PLT
movq    %rax, %rdx
...
```

Сборка программы с модулями на C++ и ассемблере

```
g++ prog1.cpp prog2.S
g++ -o prog prog1.cpp prog2.S
```



♢ Минимальная программа (*)

min.S

GNU/Linux, BSD (кроме MacOS X) 32/64; MS Windows 64

```
1 .globl main // указание компоновщику
2 main:       // начало main
3     xor %eax, %eax // 32: 0 → eax; 64: 0 → eax → rax (рек.)
4     ret // возврат из main
```

В MS Windows 32 и MacOS X любой разрядности в обоих местах _main (см. искажение имён и макрос FNAME).

Эквивалент на C++: `int main(){ return 0; }`

Целочисленное возвращаемое значение по всем соглашениям передаётся через регистр A (eax в *32, eax/rax в *64)

Сборка: **g++ min.S**

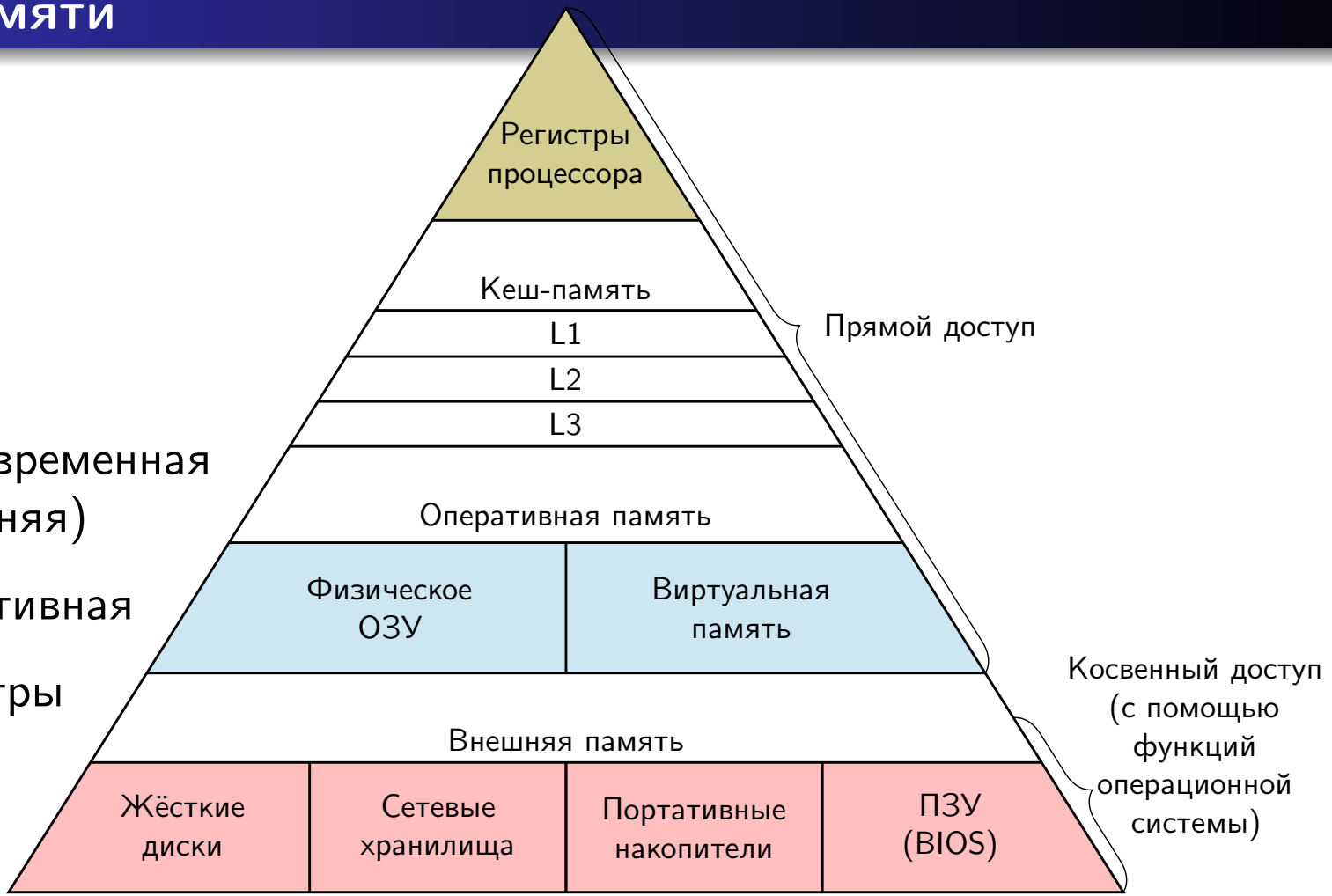
♢ Приветствие миру (GNU/Linux 64)

```
.data
    msg: .string "Hello, world!\n"
.text
.globl main
main:
    sub $8, %rsp        выравнивание sp на 16 после call main
    lea msg(%rip), %rdi } printf(msg)
    call printf
    add $8, %rsp        восстановление sp перед ret из main
    xor %eax, %eax } return 0 из main()
    ret
```

Программа не кроссплатформенна (написана для 64-разрядных GNU/Linux или BSD, кроме кроме MacOS X), так как на разных платформах используются различные соглашения о вызовах; кроме того, MacOS X искажает имена функций.

Виды памяти

- долговременная (внешняя)
- оперативная
- регистры



Прямой явный доступ: { регистры; оперативная память → «память» }

Единицы измерения памяти и коды

Элементарная ячейка памяти — **бит** (два состояния).
Машинное слово — разрядность шины данных/регистра.
Разрядность системы — в настоящее время разрядность адреса.
Байт — минимальный независимо адресуемый набор данных.
Октет — 8 бит ($2^8 = 256$ состояний).
x86: байт=октет; прямой порядок байтов (little-endian).

Беззнаковые целые числа

256 состояний $\leftrightarrow [0,255]$ — $256!$ потенциально возможных кодов

- удобство декодирования человеком;
- лёгкость построения сумматора $\rightarrow$ натуральный двоичный код

Знаковые целые числа

- натуральный код для неотрицательных;
- использование беззнакового сумматора $\rightarrow$ дополнительный код

Задачи

Переведите в двоичный код:

$0, 1, 3, 5, 23, -1, -2, -5, -7$

Рассчитайте

$3 + 5$, $(-2) + (-5)$, $23 + (-2)$, ~ 5, $!5$, $3\&5$, $3\&\&5$, $(-2)|(-5)$, $(-2)||(-5)$, $23\hat{}(-2)$

и переведите результат в десятичную форму

[Оперативная] память и регистры

1. **Память** (плоская модель) — длинный ряд пронумерованных байтов.
   - Подключается к северному (ОЗУ) и южному (разделы подкачки) мостам → относительно медленный доступ.
   - **Адрес** фрагмента памяти — номер первого байта (младший адрес).
   - Порядок байтов в простых данных (целочисленные, с плавающей запятой) определяется архитектурой:

$x =$ 0x 0A 0B 0C 0D — байты:	0	1	2	3
Прямой (little-endian, Intel, VAX), младший байт по младшему адресу	0D	0C	0B	0A
Обратный (big-endian, Motorola), младший байт по старшему адресу	0A	0B	0C	0D
Смешанный (middle-endian, mixed-endian) PDP-11	0B	0A	0D	0C
Смешанный (middle-endian, mixed-endian) Honeywell Series 16	0C	0D	0A	0B
Переключаемый (bi-endian, bytesexual)	*	*	*	*

2. **Регистры** — несколько поименованных ячеек (машинных слов).
   - Расположены непосредственно в ЦПУ → сверхбыстрый доступ.
   - Не имеют адресов, доступны только по именам.
   - Порядок байтов не определён.

В x86 байт=октет, 16 бит (8086) — слово, 32 — двойное слово, 64 — четверное.

Архитектура команд x86/x86-64: документация

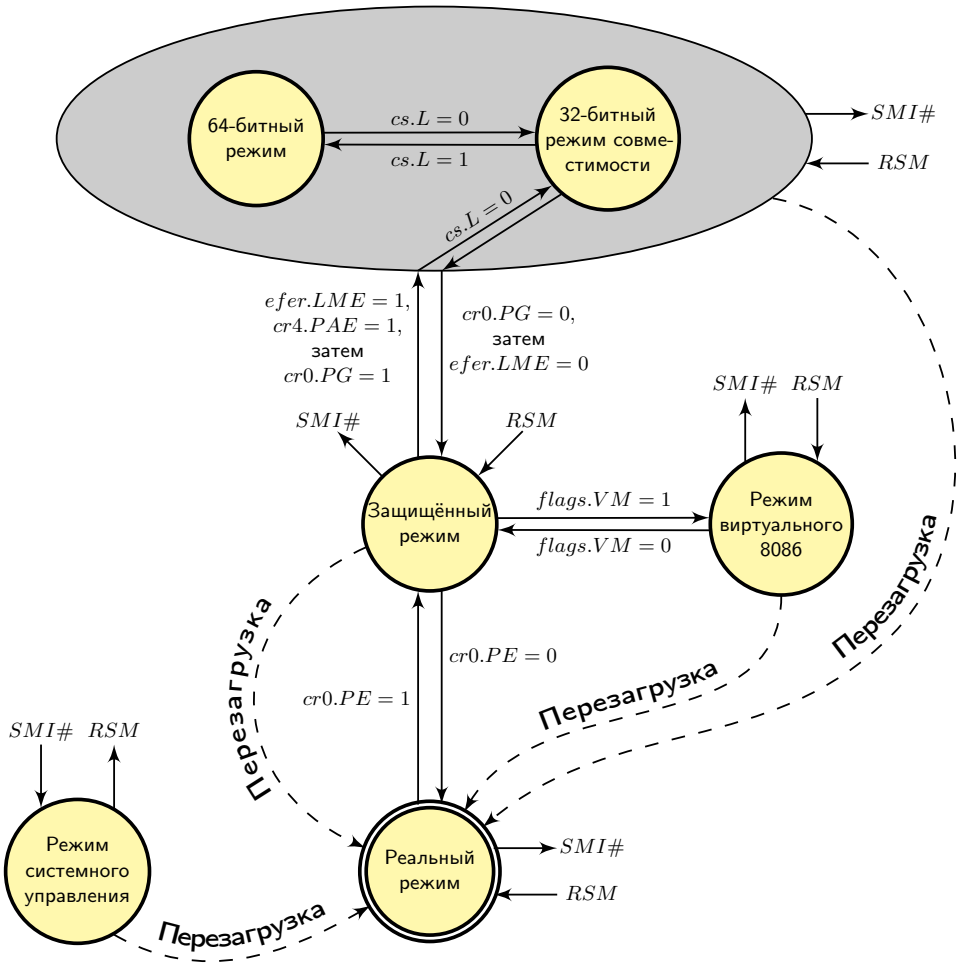
AMD, тома 1-5 одним файлом: `https://www.amd.com/en/support/tech-docs/amd64-architecture-programmers-manual-volumes-1-5`

1. прикладное программирование;
2. системное программирование;
3. команды общего назначения и системные команды;
4. AVX и SSE-команды;
5. команды FPU и MMX.

Intel, тома 1-4 одним файлом: `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html`

1. архитектура x86/x86-64;
2. полный перечень команд;
3. системное программирование;
4. моделезависимые регистры.

Режимы x86



64-битные ОС:
— основной — 64-битный режим;
— для прикладных программ доступен 32-битный режим совместимости.

32-битные ОС:
— основной — 32-битный защищённый режим;
— доступен 16-битный режим виртуального 8086.

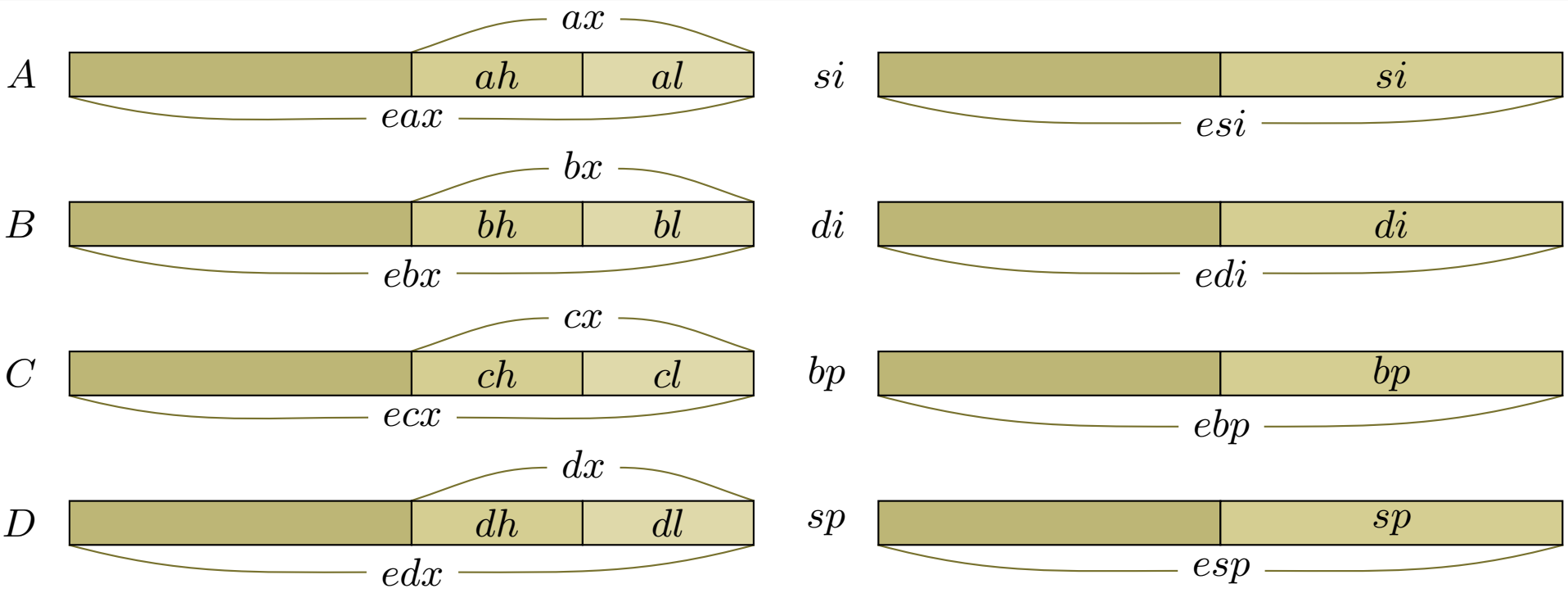
Разрядность платформы — размер адреса (в настоящее время); различия — регистры, команды, соглашения.

Регистры x86 (классы)

1. **Общего назначения, РОН** (доступны программисту целиком и по частям):
   - A $([rax/]eax/ax/al + ah)$, B, C, D;
   - si $([rsi/]esi/si[/sil])$, di, bp и указатель стека sp;

   в 64-битном режиме ещё +8 РОН:
   - $r8$ $(r8/r8d/r8w/r8b) - r11$, $r13 - r16$ и $r12$.
2. **Состояния и управления:**
   - регистр флагов $flags$ (в том числе CF, PF, AF, ZF, SF, OF);
   - указатель команды (УК) ip и т. д.
3. **Расширений**:
   - FPU (сопроцессора x87) $st(0) \ldots st(7)$/расширения MMX $mm0 \ldots mm7$
   - расширений SSE/AVX: XMM (128-разрядные $xmm0 \ldots xmm7$), YMM (256-разрядные $ymm0 \ldots ymm15$) и т. д.

РОН в 32-битном режиме



Под номер РОН в команде отводится 3 бита — 8 имён

	000	001	010	011	100	101	110	111
32-битные:	***eax***	***ecx***	***edx***	***ebx***	***esp***	***ebp***	***esi***	***edi***
16-битные:	*ax*	*cx*	*dx*	*bx*	*sp*	*bp*	*si*	*di*
8-битные:	*al*	*cl*	*dl*	*bl*	***ah***	***ch***	***dh***	***bh***

РОН в 64-битном режиме

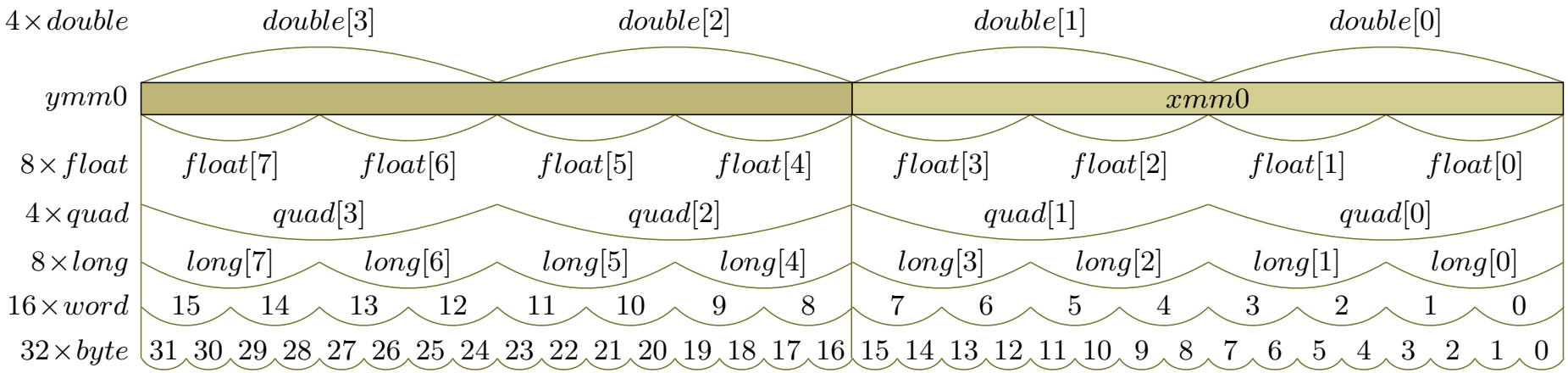
Префикс REX + команда — 16 имён (без REX 32-битный режим РОН)

	0000	0001	0010	0011	0100	0101	0110	0111	1000 . . . 1111
64-битные:	***rax***	***rcx***	***rdx***	***rbx***	***rsp***	***rbp***	***rsi***	***rdi***	***r8 . . . r15***
32-бит $(0 \to \text{hi})$:	eax	ecx	edx	ebx	esp	ebp	esi	edi	$r8d \ldots r15d$
16-битные:	ax	cx	dx	bx	sp	bp	si	di	$r8w \ldots r15w$
8-битные:	al	cl	dl	bl	***spl***	***bpl***	***sil***	***dil***	$r8b \ldots r15b$

Регистры x/ymm расширений SSE/AVX

$xmm0 = 2\times double = 2\times quad = 4\times float = 4\times long = 8\times word = 16\times byte = 128$ бит

$ymm0 = 4\times double = 4\times quad = 8\times float = 8\times long = 16\times word = 32\times byte = 256$ бит



$xmm1-xmm15$ и $ymm1-ymm15$ имеют аналогичную структуру

В 64-режиме (VEX и REX) — 16 имён ($x/ymm0-15$), в 32 — 8 ($x/ymm0-7$);
с AVX — xmm и ymm, без (очень старые либо удешевлённые) — только xmm;
с расширением AVX-512 (серверные процессоры) — xmm, ymm и $zmm0-31$.

	000	001	010	011	100	101	110	111	
$+REX/VEX$	0000	0001	0010	0011	0100	0101	0110	0111	1000 . . . 1111
256-битные:	$ymm0$	$ymm1$	$ymm2$	$ymm3$	$ymm4$	$ymm5$	$ymm6$	$ymm7$	$ymm8 \ldots ymm15$
128-битные:	$xmm0$	$xmm1$	$xmm2$	$xmm3$	$xmm4$	$xmm5$	$xmm6$	$xmm7$	$xmm8 \ldots xmm15$

Флаги ЦП

flags/eflags

0	CF	Carry Flag	Флаг переноса (беззнакового переполнения)	Состояние
1	1	—	Зарезервирован	
2	PF	Parity Flag	Флаг чётности	Состояние
3	0	—	Зарезервирован	
4	AF	Auxiliary Carry Flag	Флаг вспомогательного переноса	Состояние
5	0	—	Зарезервирован	
6	ZF	Zero Flag	Флаг нуля	Состояние
7	SF	Sign Flag	Флаг знака	Состояние
8	TF	Trap Flag	Флаг трассировки	Системный
9	IF	Interrupt Enable Flag	Флаг разрешения прерываний	Системный
10	DF	Direction Flag	Флаг направления	Управляющий
11	OF	Overflow Flag	Флаг знакового переполнения	Состояние
12–13	IOPL	I/O Privilege Level	Уровень приоритета ввода-вывода	Системный
14	NT	Nested Task	Флаг вложенности задач	Системный
15	0	—	Зарезервирован	

eflags

16	RF	Resume Flag	Флаг возобновления	Системный
17	VM	Virtual-8086 Mode	Режим виртуального процессора 8086	Системный
18	AC	Alignment Check	Проверка выравнивания	Системный
19	VIF	Virtual Interrupt Flag	Виртуальный флаг разрешения прерывания	Системный
20	VIP	Virtual Interrupt Pending	Ожидающее виртуальное прерывание	Системный
21	ID	ID Flag	Проверка на доступность инструкции CPUID	Системный
22–31		—	Зарезервированы	

Архитектура команд и ассемблер

2. **Архитектура команд** — регистры и память, адресация, набор команд:
   - x86 — CISC *(complex instruction set computer)*

↑ загрузка и запуск ↕ системные вызовы (интерпретация)

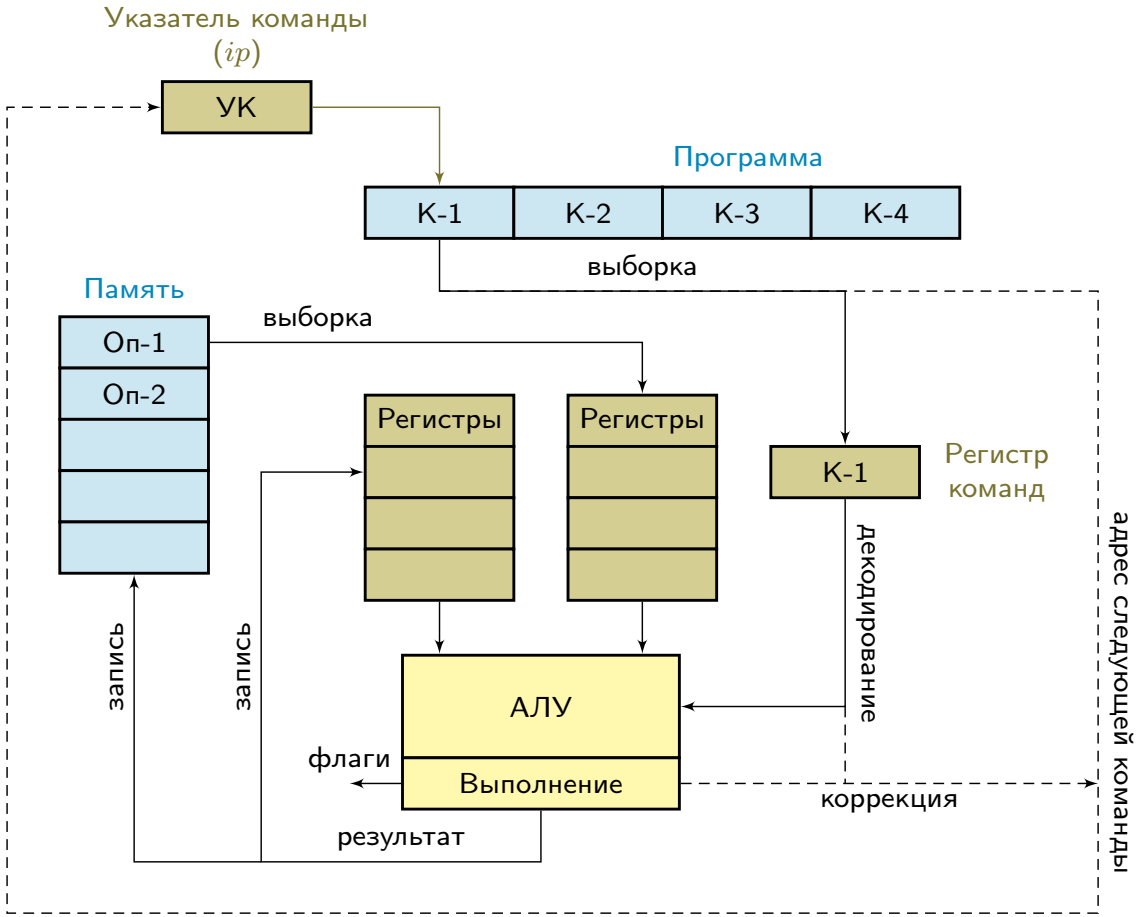
3. **Операционная система**

↑ компиляция в исполняемый файл

```
B9 CC CC CC CC
      ↑
movl $0xCCCCCCCC, %ecx
```

4. **Язык ассемблера** — мнемоники, директивы, синтаксис:
   - Intel
     - множество несовместимых диалектов;
     - доступен только для Intel-подобных компьютеров;
   - **AT&T** (GAS) — GNU Assembler (коллекция GCC).

Интерпретация программы: арх. команд → микроархитектура



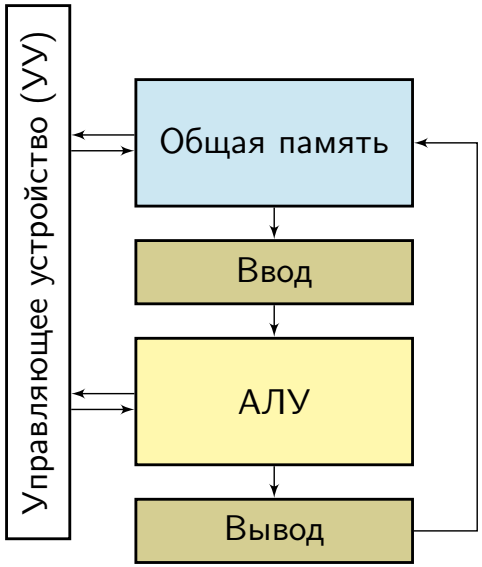
Конвейер:

- выборка и декодирование команды (адр. след.)
- выборка операндов
- выполнение команды
- запись результатов и установка **флагов**
- окончательное формирование адреса следующей команды

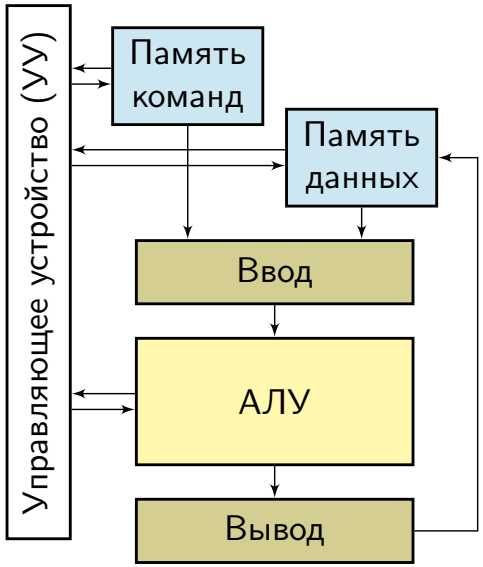
$\times N \rightarrow$ суперскалярн.

Память, программы и данные

1. Программы.
2. Данные:
   - глобальные и статические переменные;
   - локальные переменные функций;
   - динамические переменные.



Фон-неймановская (принстонская) архитектура



Гарвардская архитектура

Общая шина для памяти программ и данных — узкое место.
Современные x86 — общая шина, но раздельные кеши программ и данных.

Разделы памяти

Text — код программы
Data — глобальные и статические переменные с начальным значением
BSS — неинициализированные глобальные и статические переменные

↑ статически, компилятор

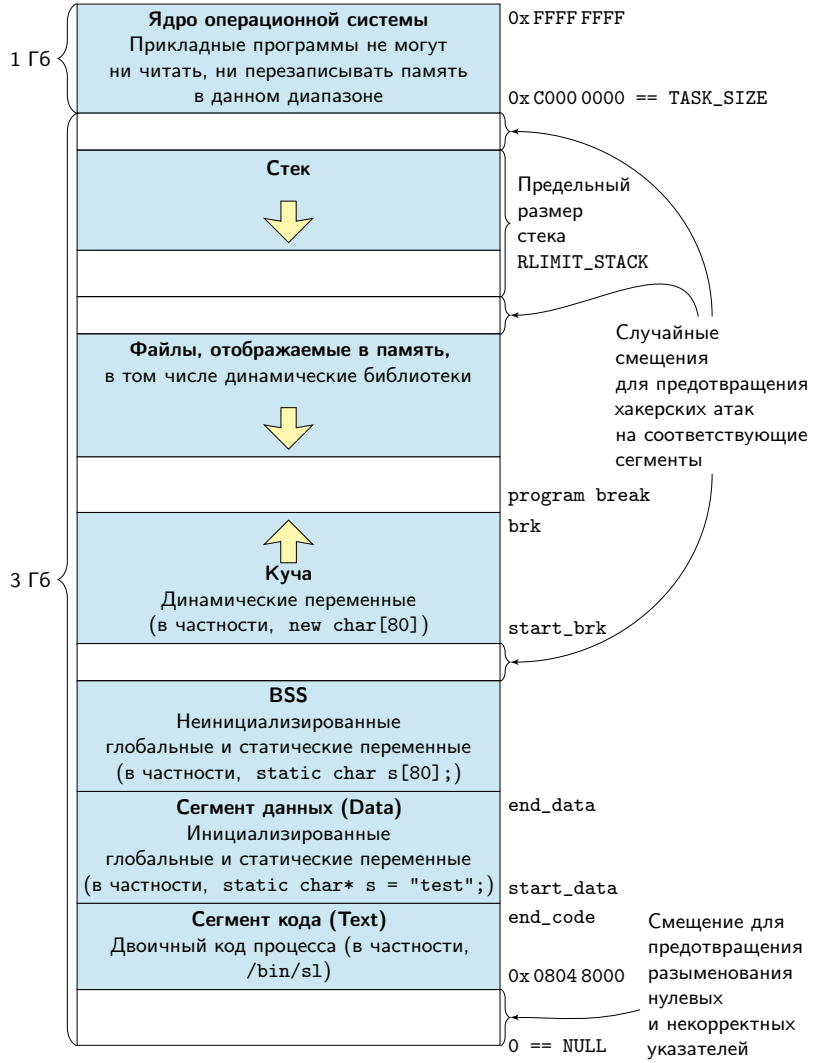
Куча (динамическая память) — new/delete, *alloc()/free()

↑ динамически, системные вызовы или их обёртки

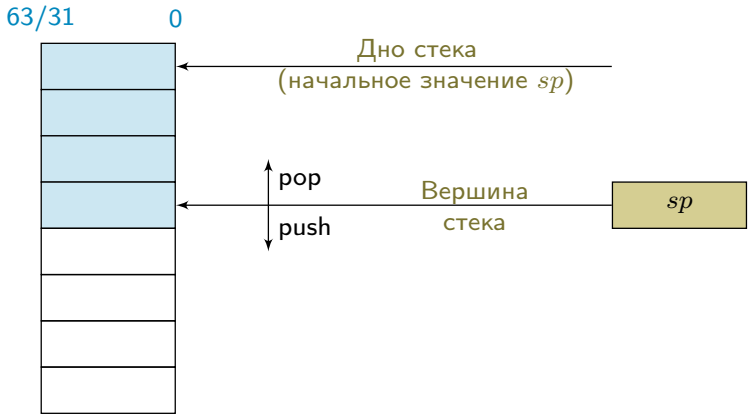
Стек — данные функций (локальные переменные, параметры, служебные)

↑ динамически, указатель вершины стека sp (rsp/esp)

Показано распределение памяти в 32-битной GNU/Linux

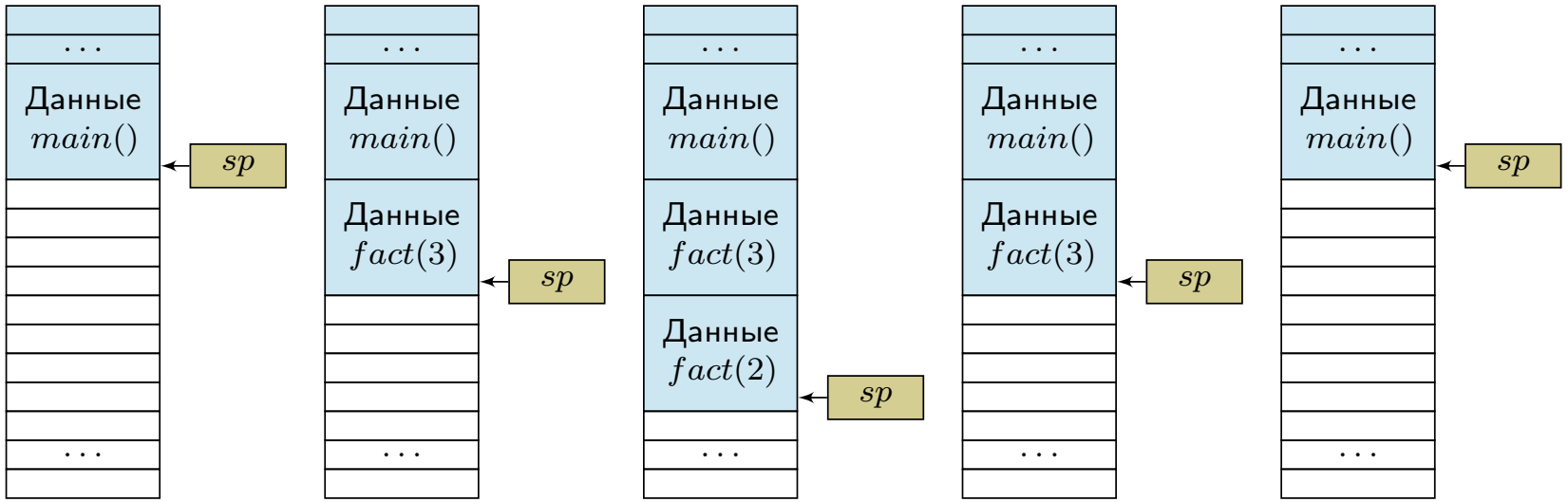


Стек

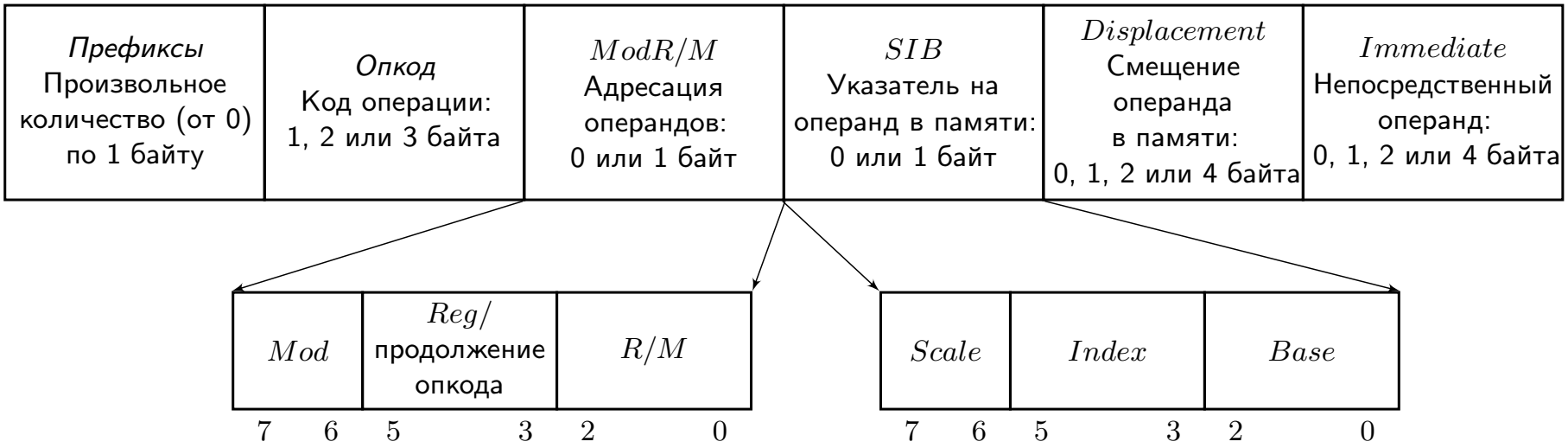


Стек содержит данные функций — параметры, адреса возврата и локальные переменные

Рассмотрим рекурсивное вычисление факториала $fact(3)$:



Команда x86 и её операнды (src/dst)



1. Reg — регистр (РОН).
2. R/M — РОН (при $Mod = 11$) или указывает на адрес в памяти:
   - $Base + 2^{Scale} \cdot Index + Displacement$,
     где $Base$ и $Index$ — РОН ($Index$ не может быть sp и $r12$) или 0;
   - $rip + Displacement$ (только в 64-битном режиме, $\begin{cases} Mod = 00 \\ R/M = 101 \end{cases}$).
3. $Immediate$ — непосредственный операнд (константа, магическое число).

0. Регистр в опкоде.

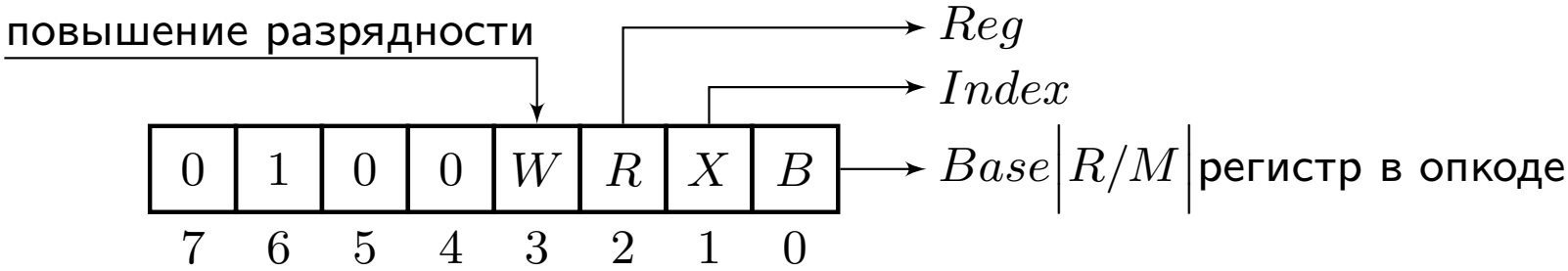
Методы адресации

1. **Неявная** (единственный способ адресовать специальные регистры).
2. **Непосредственная** — константа, прошитая в команде ($Immediate$).
3. **Регистровая** — переменная в регистре (поля Reg и R/M).
4. **Прямая относительная** — переменная в памяти по фиксированному адресу $Addr$, в команду включается смещение относительно указателя команд ip: $Displacement = Addr - ip$.
5. **Прямая** (абсолютная) — переменная в памяти по фиксированному прошитому в команде адресу $Addr$ (статическая): $Displacement = Addr$ без базы и индекса.
6. **Косвенно-регистровая (косвенная)** — переменная в памяти по адресу
$$Base + 2^{Scale} \cdot Index + Displacement,$$
где $Base$ и $Index$ — регистры, $Scale \in \{0, 1, 2, 3\}$ и $Displacement$ — константы, любая компонента может быть опущена.

Размер операндов

Операнды-данные:

- **32 бита** по умолчанию в 32-битном и 64-битном режимах (кроме **push/pop**, где в 64-битном режиме 64 и не понижается до 32);
- понижение до 16 бит — префикс 0x66;
- повышение до 64 бит — префикс REX с битом $REX.W = 1$:



- **8 бит** — отдельные команды!

Операнды-адреса (неявный sp, jmp/jCC/call/ret, $Base$, $Index$):

- **64 бита** в 64-битном режиме и **32 бита** в 32-битном по умолчанию;
- понижение — префикс 0x67 (до 16 в 32-битном режиме; в 64-битном: jmp/jCC/call/ret — до 16; $Base$, $Index$ — до 32).

Спасибо за внимание!

МИЭТ
www.miet.ru

Александра Игоревна Кононова / **illinc@mail.ru**
+7-985-148-32-64 (телефон), +7-977-977-97-29 (WhatsApp),
gitlab.com/illinc/raspisanie