Введение

Основы информатики

Компьютерные основы программирования

На основе CMU 15-213/18-243:
Introduction to Computer Systems

Лекция 1, 04 февраля, 2025

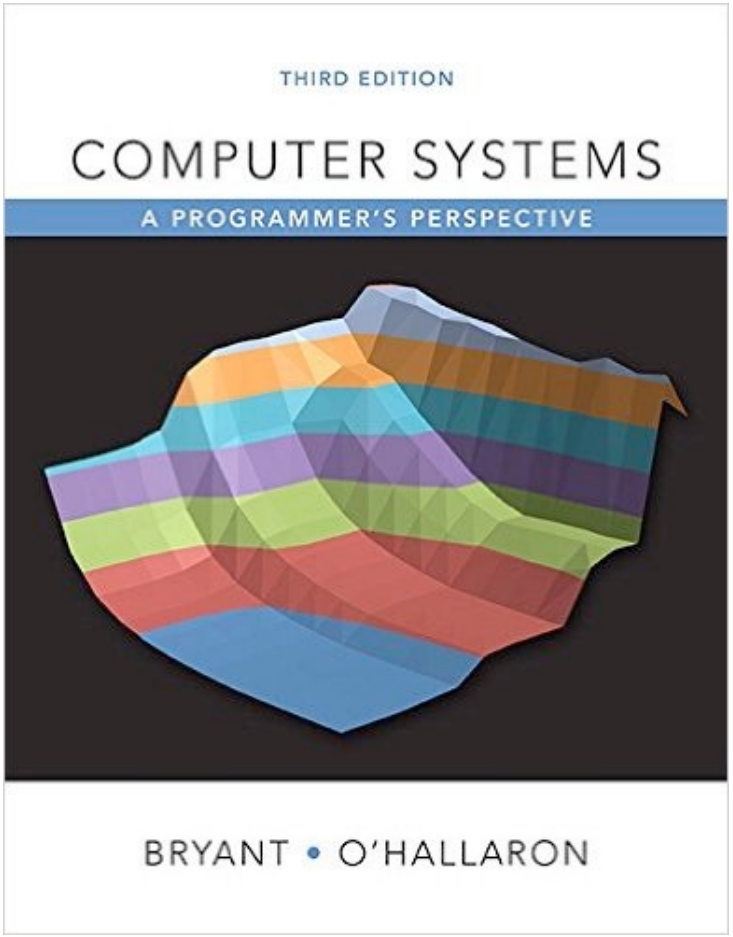
Лектор:

Дмитрий Северов, кафедра информатики 608 КПМ

Литература

- Randal E. Bryant and David R. O'Hallaron,
  - "Computer Systems: A Programmer's Perspective, Third Edition" (CS:APP3e),
  - http://csapp.cs.cmu.edu
- **Взгляд программиста**
  - Брайант Р. , О`Халларон Д. Компьютерные системы: архитектура и программирование. 3-е издание ISBN 978-5-97060-492-2
- **Учебник по Ассемблеру**
  - Ирвин, Кип. Язык ассемблера для процессоров Intel, 4е издание. ISBN 5-8459-0779-9
- **Справочник по Ассемблеру**
  - Юров В.И., Assembler: Специальный справочник. ISBN 5-469-00003-6
- **Учебник по архитектуре**
  - Э. Таненбаум Архитектура компьютера
    ISBN 5-469-01274-3

Первоисточники



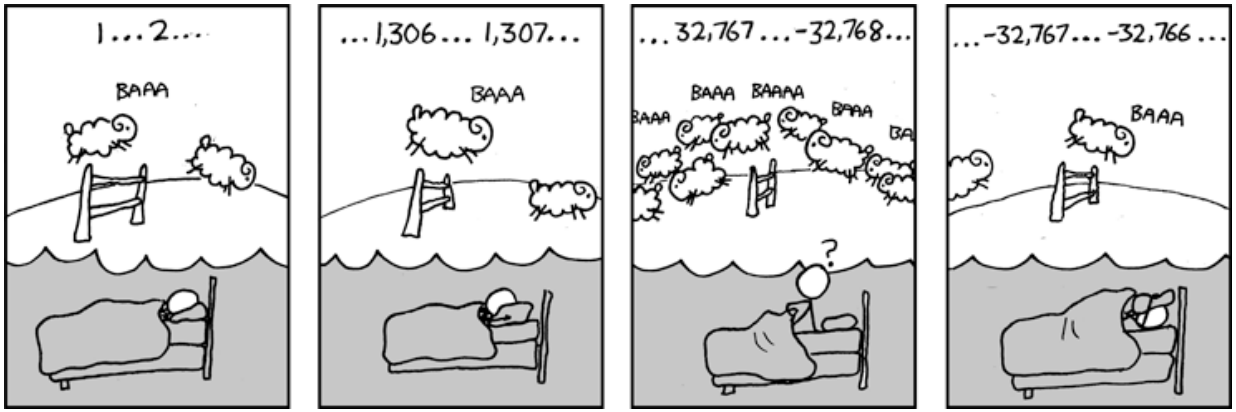
- http://www.cs.cmu.edu/afs/cs/academic/class/15213-f15/www/index.html

Идея курса:
Абстракции хороши, но знайте страшную правду

- 1-й семестр посвящён абстракциям
  - Абстрактные типы данных
  - Асимптотический анализ производительности
- Абстракции имеют ограниченное применение
  - Особенно при наличии ошибок
  - Требуют понимания деталей реализации
- Предполагаемая польза данного курса:
  - Ваш личный рост как эффективного программиста
    - Способность эффективно находить и устранять ошибки
    - Способность улучшать быстродействие программ
  - Подготовка к следующим компьютерным курсам
    - Операционные системы, сети, компиляторы,…

Страшная правда №1: int-ы не целые, float-ы не вещественные

- Пример 1: $x^2 \geq 0$?
  - Для float-ов: Да!



  - Для int-ов:
    - 40000 * 40000 → 1600000000
    - 50000 * 50000 → ??
- Пример 2: (x + y) + z = x + (y + z) ?
  - Для unsigned и signed int-ов: Да!
  - Для float-ов:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Арифметика компьютера

- **Не создаёт случайных значений**
  - У арифметических операций есть важные математические свойства
- **Не предполагает всех “обычных” математических свойств**
  - Из-за ограниченности представления
  - Целочисленные операции удовлетворяют свойствам «кольца»
    - Коммутативность, ассоциативность, дистрибутивность
  - Операции с плавающей точкой удовлетворяют свойству «упорядоченности»
    - Монотонность, знаки
- **Наблюдение**
  - Необходимо понимать в каком случае какие абстракции применять
  - Важно для разработчиков компиляторов и ответственных приложений

Страшная правда №2:
Вам полезно знать ассемблер

- Вероятно, вы никогда не будете программировать на ассемблере
  - Компиляторы много эффективнее и терпеливее вас
- Но: Понимание ассемблера – ключ к модели исполнения машинного уровня
  - Поведение программ при наличии ошибок
    - Модели высокоуровневых языков – терпят неудачу
  - Улучшение быстродействия программ
    - Понимание оптимизаций [не]сделанных компилятором
    - Понимание источников неэффективности программ
  - Реализация системного ПО
    - Цель компилятора – машинный код
    - Операционная система управляет состоянием процесса
  - Создание вредоносного кода и противодействие ему
    - ассемблер x86 для ПК, ARM для мобильных, MIPS для IoT – то что надо!

Пример ассемблерного кода

- Счётчик метки времени
  - Специальный 64-битовый регистр в Intel-совместимых машинах
  - Увеличивается с каждым циклом
  - Считывается командой `rdtsc`
- Применение
  - Измерение времени (в циклах) требуемого программой

```
double t;
start_counter();
P();
t = get_counter();
printf("P требует %f циклов\n", t);
```

Код читающий счётчик

- Напишем немного ассемблерного кода, используя возможность ассемблерной вставки
- Ассемблерный код вставляется в машинный код, порождаемый компилятором

```
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;

/* Поместить в *hi and *lo старшие и младшие биты счётчика
циклов.
*/
void access_counter(unsigned *hi, unsigned *lo)
{
    asm("rdtsc; movl %%edx,%0; movl %%eax,%1"
        : "=r" (*hi), "=r" (*lo)
        :
        : "%edx", "%eax");
}
```

Страшная правда №3: Память имеет значение, ОЗУ(RAM) – далёкая от реальности абстракция

- Память не является неограниченной
  - Память требует размещения и управления
  - Во множестве приложений доминирует память
- Ошибки ссылок на фрагменты памяти – самые вредные
  - Эффекты отстоят от причины во времени и пространстве
- Быстродействие памяти не однородно
  - Эффекты кэша и виртуальной памяти значительно влияют на быстродействие
  - Адаптация программы к характеристикам подсистемы памяти – основной резерв улучшения быстродействия

Пример ошибки обращения к памяти

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Возможно за границами допустимого */
  return s.d;
}
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.00000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

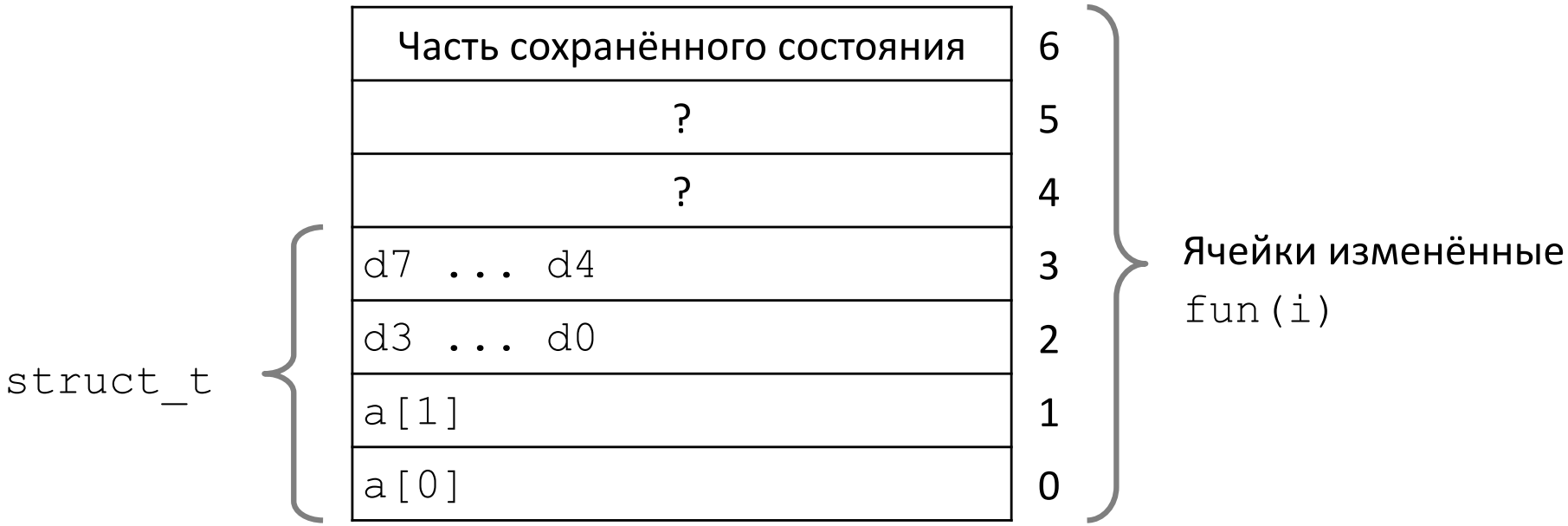
- Результат зависит от архитектуры

Пример ошибки обращения к памяти

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.0000000610351 56
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

Пояснение:



Ошибки ссылок в память

- C и C++ не обеспечивают защиты памяти
  - Выход за границы массивов
  - Негодные значения указателей
  - Некорректное использование malloc/free
- Могут приводить к скверным последствиям
  - Эффект ошибки проявляется в зависимости от системы и компилятора
  - Отложенное действие
    - Повреждённый объект логически не связан с тем, к которому выполняется доступ
    - Эффект ошибки может наблюдаться много позже возникновения
- Что с этим делать?
  - Программировать на Rust, Java, Ruby, Scala, ML, …
  - Понимать, какие возможные взаимодействия могут возникнуть
  - Использовать или создавать средства обнаружения (напр. Valgrind)

Страшная правда №4: Быстродействие – не сводится к асимптотической сложности

- Постоянные множители тоже имеют значение!
- И даже точный подсчёт операций не предсказывает быстродействия
  - Легко увидеть 10-кратную разницу в зависимости от написания кода
  - Оптимизация должна производиться на нескольких уровнях : алгоритм, представление данных, процедуры, циклы
- Для оптимизации необходимо понимать систему
  - Как программы компилируются и исполняются
  - Как измерять быстродействие программ и обнаруживать узкие места
  - Как улучшать быстродействие не нарушая модульности и общности кода

Пример производительности памяти

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

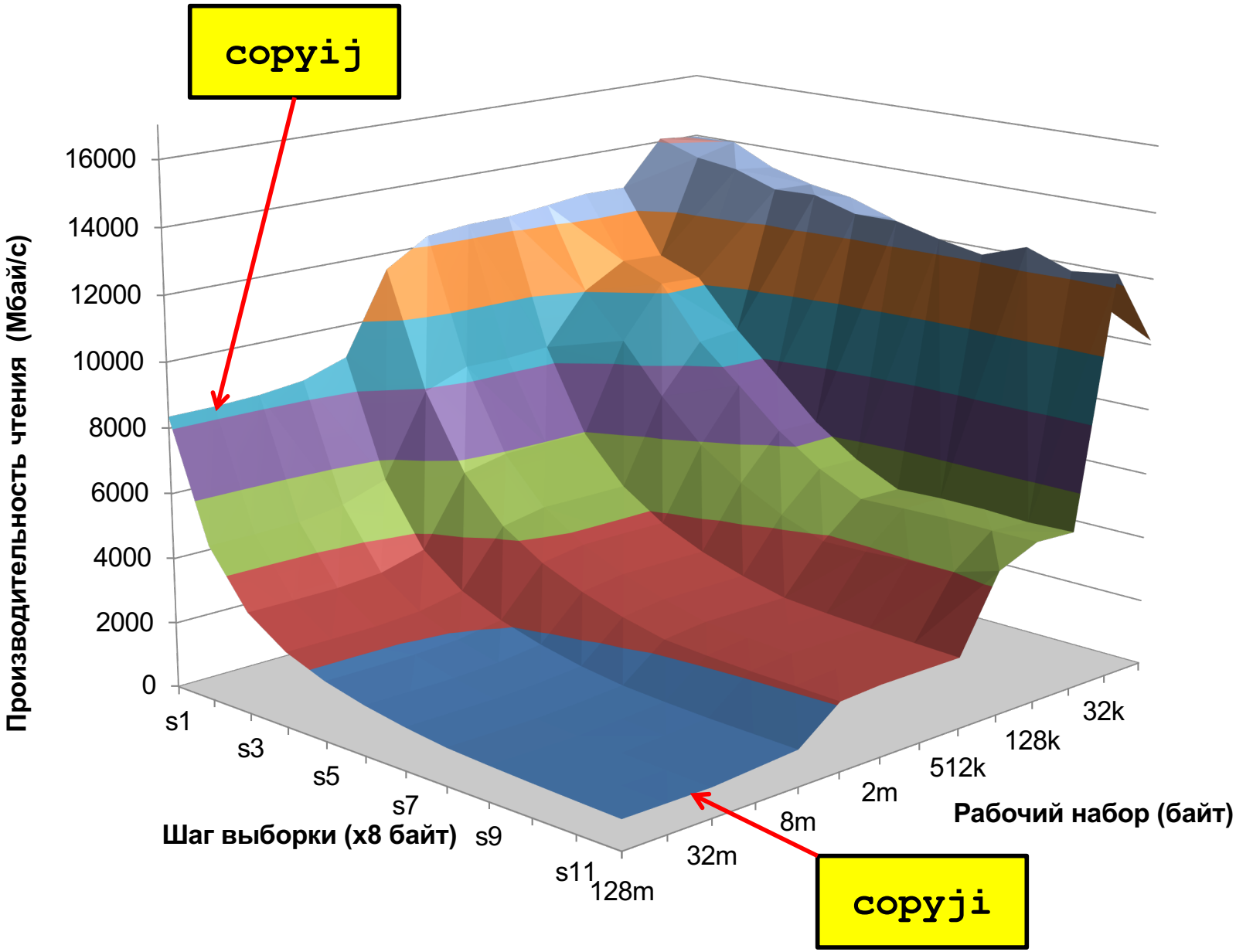
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms 81.8ms

2.0 GHz Intel Core i7 Haswell

- Иерархическая организация памяти
- Производительность зависит от организации доступа
  - включая способ перебора многомерного массива

Почему производительность разная ?



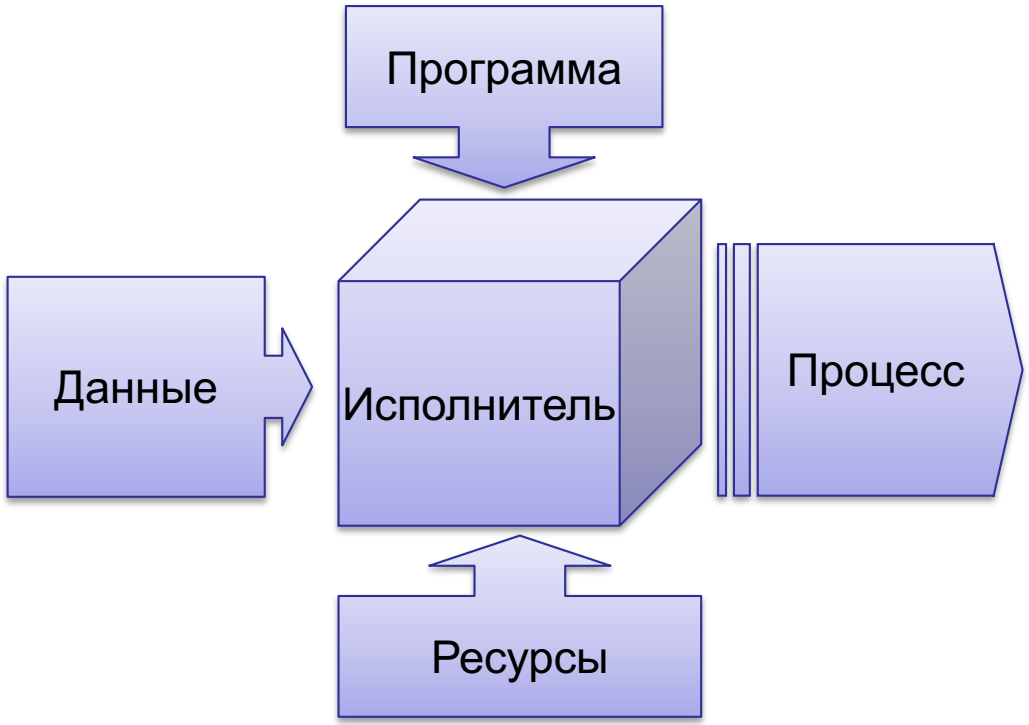
Страшная правда №5: Компьютеры не только исполняют программы

- Компьютеры транслируют и собирают программы
- Компьютеры вводят и выводят данные
  - Подсистема ввода/вывода критически значима для надёжности и быстродействия программ
- Компьютеры взаимодействуют друг с другом в сети
  - Многие проблемы системного уровня порождаются сетью
    - Параллельное выполнение автономных процессов
    - Копирование на ненадёжный носитель
    - Межплатформенная совместимость
    - Сложные проблемы производительности

Картина курса в целом

- Курс помещает программиста в «центр картины»
  - Цель – показать как стать более эффективным программистом зная больше об используемой системе
  - Даёт вам возможность
    - Создавать более надёжные и эффективные программы
  - Это не курс для посвятивших себя хакерству
    - Хакер сидит в каждом

Понятия программируемых действий



Программа

определённый набор предписаний

Данные

неопределённый набор бит

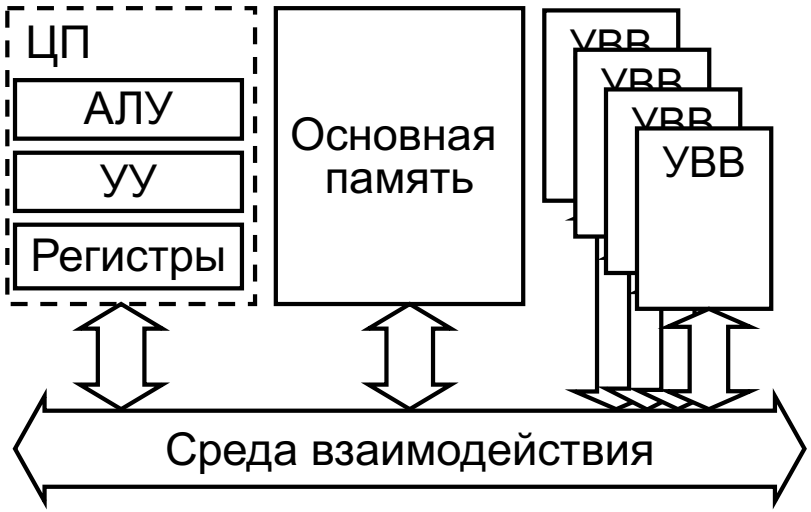
Процесс

ожидаемый набор **фактических** событий

Ресурсы

- времени
- энергии
- пространства

Масштабы событий



Программа

- **Ваш процесс**
  - **Выполнение команд**
  - **Изменение состояния**
- Другие процессы

Крупнее: среда исполнения

- Загрузка кода программы
- Выделение ресурсов
- Нормирование работы и реагирование на запросы
- Освобождение ресурсов

Мельче: команда

- Выборка кода операции
- Декодирование операции
- Выборка операндов
- Выполнение операции
- Сохранение результата

Создание программы

Редактирование

⇓ Текст программы

Трансляция

⇓ Объектный код

Компоновка

⇓ Загрузочный код

Загрузка

⇓ Исполняемый код

Выполнение, отладка

⇓ Результат

Создание программы вместе

Редактирование

⇓ Текст программы ⇐ Ваши изменения вручную

Трансляция

⇓ Объектный код ⇐ Библиотечный исходный текст

Компоновка

⇓ Загрузочный код ⇐ Библиотечный машинный код

Загрузка

⇓ Исполняемый код ⇐ Решения среды исполнения

Выполнение, отладка

⇓ Результат ⇐ Внешние данные, коды, события

Создание программы в окружении

Редактирование	☝ **Предписания редактору**
⇓ Текст программы	⇐ Ваши изменения вручную
Трансляция	☜ Предписания трансляции
⇓ Объектный код	⇐ Библиотечный исходный текст
Компоновка	☜ Предписания компоновки
⇓ Загрузочный код	⇐ Библиотечный машинный код
Загрузка	☜ Предписания загрузки
⇓ Исполняемый код	⇐ Решения среды исполнения
Выполнение, отладка	☜ Предписания исполнения
⇓ Результат	⇐ Внешние данные, коды, события

Алгоритм

- Определение
  - **ГОСТ 33707-2016:** Конечное упорядоченное множество точно определенных правил для решения конкретной задачи.
  - **ISO 2382/1-84:** Конечный набор предписаний, определяющий решение задачи посредством конечного количества операций
- Свойства
  1. Дискретность: данных и действий над ними
  2. Понятность: доступность и однозначность предписаний
  3. Конечность: решение задачи - за конечное число шагов
  4. Определённость: воспроизводимость результата
  5. Массовость: применимость к различным данным
- Представление - программа на формальном языке
  - Действия: операции и операторы
  - Данные: типы и экземпляры
  - Организация: конструкции сложных данных и действий
  - Окружение: взаимодействие со средой выполнения

Модели, языки, машины

- **Модель** – упрощённое представление поведения частей и отношений системы
- **Язык** - знаковая подсистема фиксации, переработки и передачи информации о модели.
- **Машина** – исполнитель, выполняющий предписания языка.

- Рынки
- Потребители
- Задачи

- Алгоритмы
- Программы
- Система/Аппаратура
- Цифровые схемы
- Аналоговые схемы

- Физические явления в [не]линейных структурах

Модели, языки, машины

- **Модель** – упрощённое представление поведения частей и отношений системы

- Рынки
- Потребители
- Задачи

…
язык ассемблера

язык архитектуры набора команд
…

- Алгоритмы
- Программы
- Система/Аппаратура
- Цифровые схемы
- Аналоговые схемы

- Физические явления в [не]линейных структурах

… среди компьютерных языков

■ Алгоритмы	■ Языки спецификаций
■ Программы	■ Языки высокого уровня ■ **Язык ассемблера**
■ Архитектура системы ■ Архитектура аппаратуры	■ Язык системных вызовов ■ Язык аппаратурных команд
■ Блоки архитектуры ■ Цифровые схемы ■ Аналоговые схемы	■ Язык микрокоманд ■ Язык цифровых схем ■ Язык аналоговых схем

(Само)обман

- Что есть (само)обман ?
  - Брать чужой код: копируя, перенабирая, **подглядывая**, принимая файлы
  - Пересказывать: устное описание кода одним человеком другому
  - Натаскивание: помощь другу в построчном написании заданий
  - Поиск решения в сети
  - Копирование кода предыдущих кусов и экзаменов

- Что НЕ есть (само)обман?
  - Объяснять как использовать инструменты или системы
  - Помогать другим в вопросах конструкции верхнего уровня

Рекомендации к практической работе

- Запустить примеры на C (на слайдах №)
  - со счётчиком машинных циклов (8-9)
  - с ошибкой обращения в память (11-12)
  - с перестановкой индексов (15)
- Найти способ выдачи ассемблерного листинга