ФРТК

iSCALARE

intel

Лаборатория суперкомпьютерных технологий для биомедицины, фармакологии и малоразмерных структур

Моделирование центрального процессора с помощью интерпретации

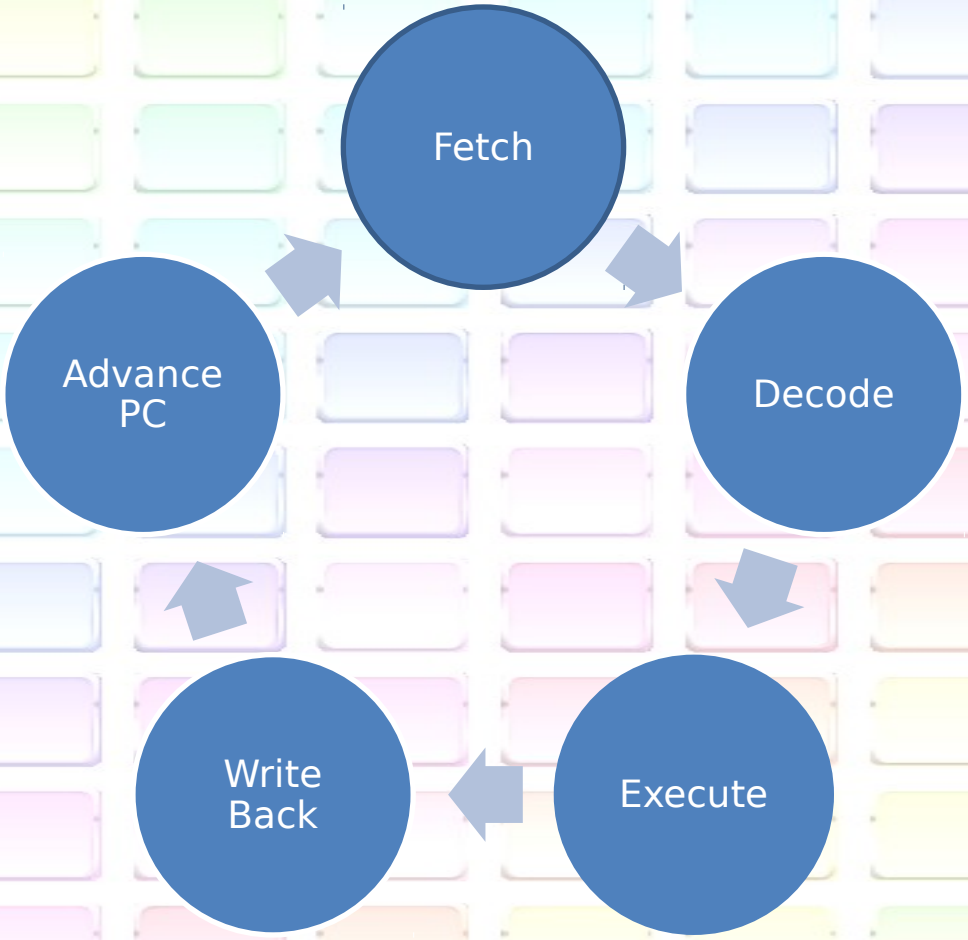
Григорий Речистов

grigory.rechistov@phystech.edu

18.02.2013

- Общий цикл работы процессора ↔ цикл интерпретации
- Эмуляция одной инструкции
- Проблемы интерпретации
- Улучшенные схемы

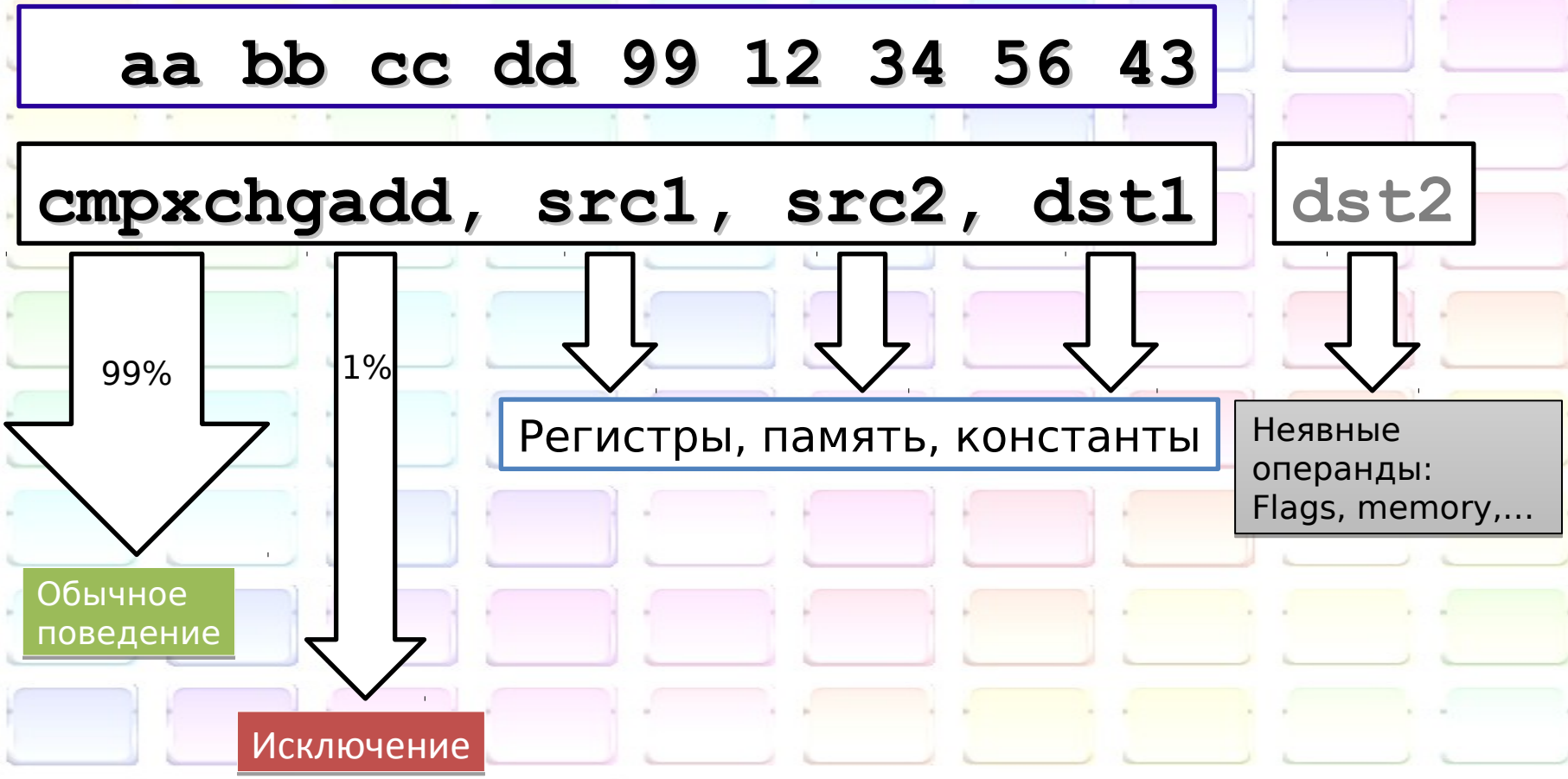
Основной цикл работы процессора



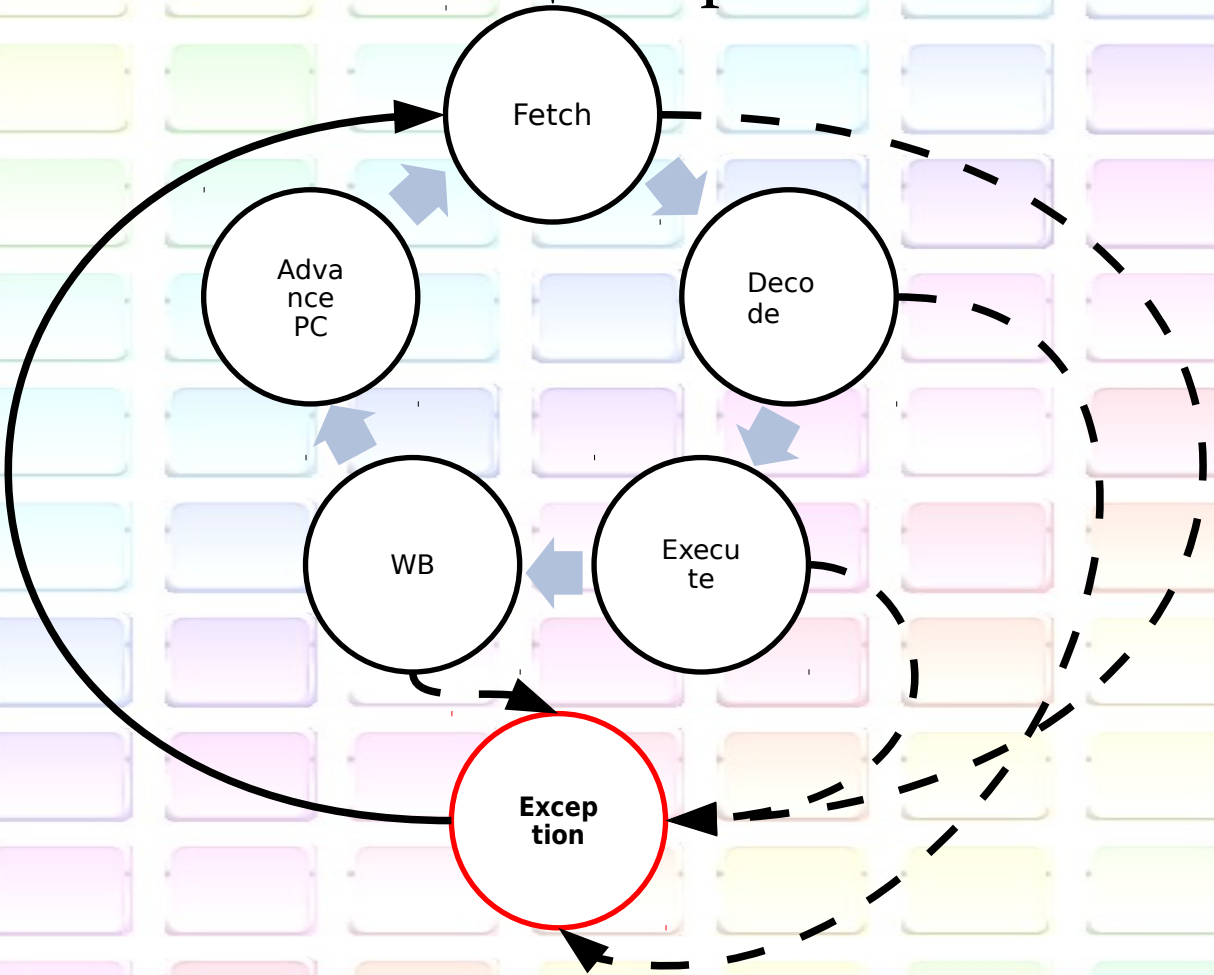
Код (switched, переключаемый)

```
while (!interrupt) {
  raw_code = fetch(PC);
  (opcode, operands) = decode(raw_code);
  switch (opcode) {
    case opcode1: func1(operands); PC++; break;
    case opcode2: func2(operands); PC++; break;
     /*...*/
  }
}
```

Анатомия одной инструкции



Уточнённый цикл работы



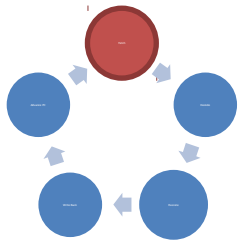
Исключительные ситуации

Interruptions (термин из документации IA-64) — вмешательство (?)

- **Exception** — синхронное исключение, без повторения текущей инструкции
- **Fault** — синхронное, с повторением текущей инструкции
- **Trap** — синхронные, без повторения, «намеренно» вызывающие исключение
- **Interrupt** — внешнее асинхронное прерывание

Чтение инструкции из памяти

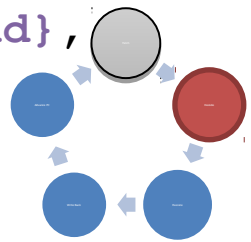
- «Простое» чтение байт из памяти?
  - Невыровненный (unaligned) адрес в памяти
    - Вызывает забавные эффекты в некоторых архитектурах
  - Доступ на границе двух страниц памяти
    - Разные страницы могут иметь разные характеристики



Декодирование (1/3)

- Перевод данных об инструкции из машинного представления во внутреннее (высокоуровневое), удобное для последующего анализа
- Вход: “df ce 0f ad de”
- Результат:

```
instruction {
  opcode = ADDL_R_M, num_operands = 2,
  src1 = {type = OP_REG, length = 32, reg = R15, },
  dst2 = {type = OP_MEM, length = 16, offset = 0xdead},
  disasm = “addl %r15, (0xdead)”,
  address = 0x11002233
}
```



Декодирование (2/3)

- Код декодера редко пишется вручную, он генерируется по описаниям:

```
шаблоны                  => интерпретация

110011XXXYYYY__ => sqrtpd X Y
```

- В общем случае – классическая задача построения синтаксического анализатора
- Пример декодера – XED (X86 encoder-decoder)
  - **http://www.pintool.org/docs/24110/Xed/html**

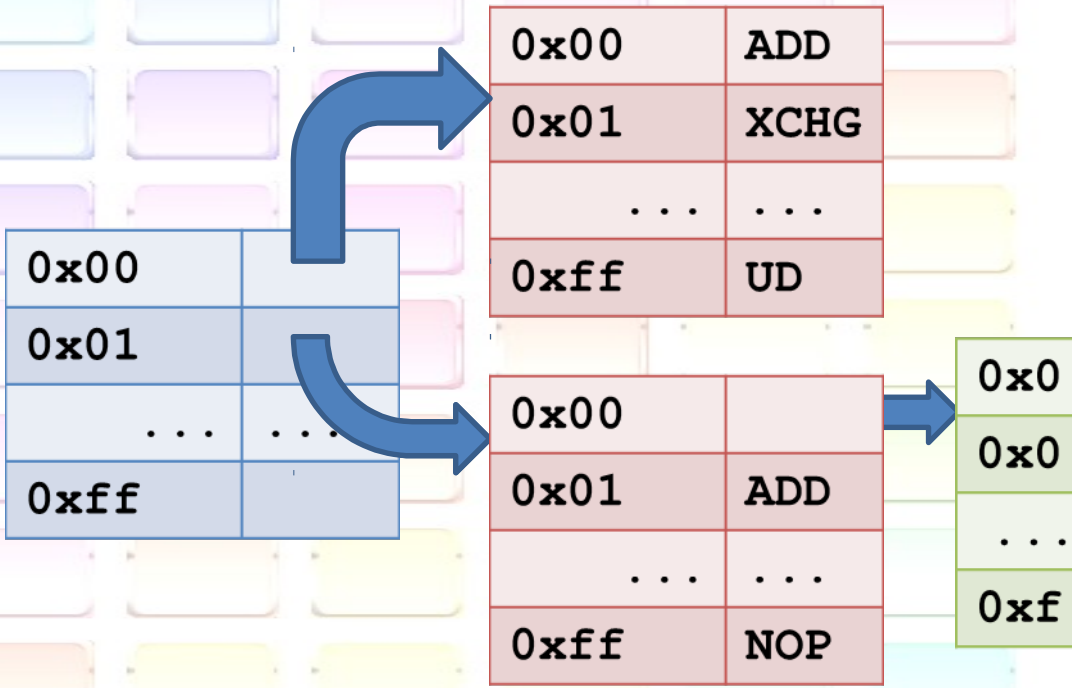
Декодирование (3/3)

Таблица опкодов

(8, 16 битные архитектуры)

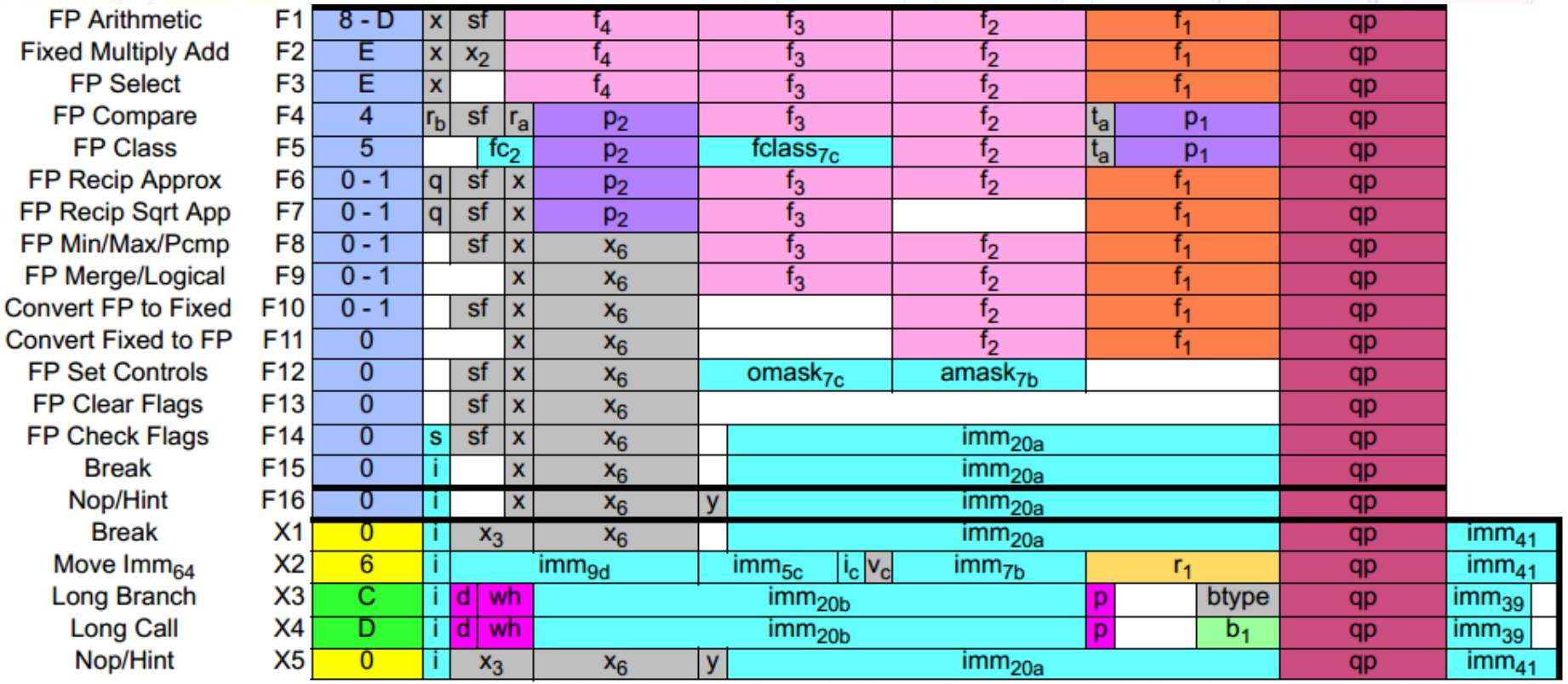
0x0000	ADD
0x0001	SUB
0x0002	MUL
...	...
0xffff	NARF

Таблицы префиксных кодов



Пример: Itanium 2.3

Intel® Itanium® Architecture Software Developer's Manual, p 3:296



Дизассемблирование

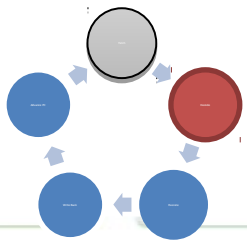
- Перевод инструкции из машинного представления в текстовый вид, понятный для человека (в мнемонику)

Закодирование (encoding)

- Перевод инструкции из мнемонической записи в машинный код
- Перевод инструкции из декодированной структуры в машинный код

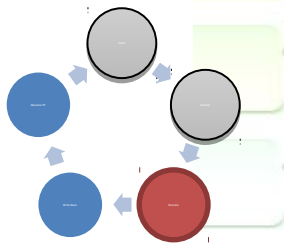
Декодирование: суровая реальность

- Переменная длина инструкций
  - IA-32: от 8 до 128 бит
  - Сколько байт пытаться декодировать за один раз?
- Не префиксный код
- Зависимость смысла от префикса, режима работы процессора
  - Пример: **`0x48`** в IA-32
- Полное несоответствие какому-либо здравому смыслу



Исполнение

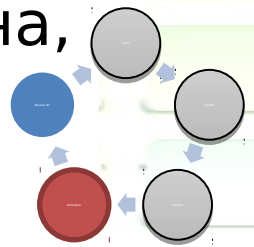
- Базовая единица — функция-эмулятор одной инструкции (service routine)
- S.r. пишутся на языке высокого уровня – переносимость между хозяйскими платформами, компиляторами
- Используются генераторы кода по описанию
  - Пример: SimGen — из одного описания совмещает генерируются декодер, дизассемблер и s.r.



Запись результата в память

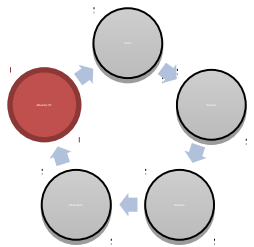
«Обычная» запись в память

- Невыровненный адрес
- Граница страниц
- Попытка изменить регион памяти «только для чтения»
- Часть результата может быть записана, а затем случится исключение



Продвижение **$PC**

- Для большинства команд – увеличение счётчика на длину обработанной инструкции
  - Пример исключения из правил: **`REP MOVS`**
- Явное изменение PC - команды управления исполнением:
  - (Un)conditional (In)direct Jump/Branch
  - Call/Return (subroutine)
  - Исключения

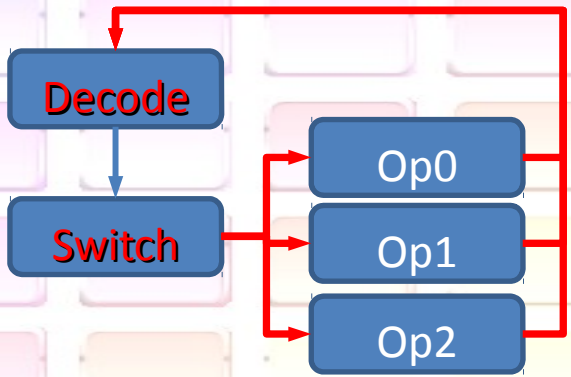


За и против И.

- Пишется на «нормальном» языке – код переносим и читабелен
- Простая структура
  - Надёжность
  - Расширяемость
  - Возможность переиспользования кода
- Скорость работы: от невысокой до совсем черепашьей

Куда тратится время?

```
while (!interruption) {
raw_code = fetch(PC);
(opcode, operands) = decode(raw_code);
        switch opcode {
        case opcode1: func1(operands);
                PC++; break;
        case opcode2: func2(operands);
                PC++; break;
            ...
        }
}
```



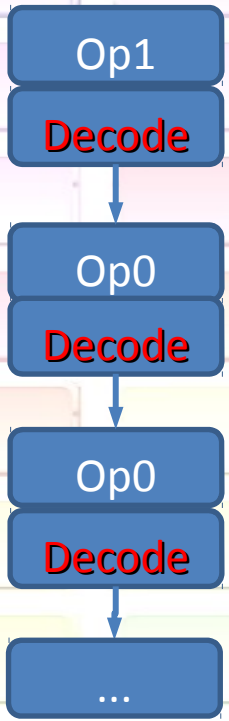
Как ускорить интерпретацию?

- Избавляемся от медленных операций
- Не делаем то, что уже сделано.
  - Поиск в таблицах => обращение в память
  - Декодирование (== поиск в таблицах)
- Используем ресурсы хозяина эффективно
  - Предсказатель переходов, обращения в память

Сцепленная интерпретация (Threaded interpretation)

Вместо возвращения к началу цикла «прыгаем» прямо на исполнение следующей инструкции

```
func0:
  /* simulate instr0 */; PC++;
  next_opcode = decode(fetch(PC));
  goto func_ptr[next_opcode];
func1:
  /* simulate instr1 */; PC++;
  next_opcode = decode(fetch(PC));
  goto func_ptr[next_opcode];
func2:
  /* simulate instr2 */; PC++;
  next_opcode = decode(fetch(PC));
  goto func_ptr[next_opcode];
```



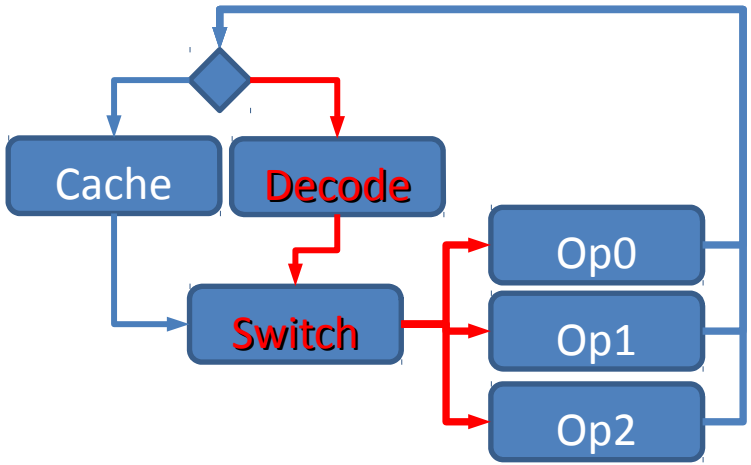
Кэширующая интерпретация (1/3)

- В большинстве случаев в код гостевого приложения неизменен
- Велика вероятность того, что инструкции с некоторыми **`$PC`** будут исполнены много раз
- Зачем каждый раз декодировать их?
- Заводим таблицу

`addr => decoded_instruction`

Кэширующая интерпретация (2/3)

```
while (true) {
  if (operation = cache[PC]); // shortcut
  else { // not in cache, long way
    operation = decode(fetch(PC));
    cache[PC] = operation;
  }
  switch (operation) {
     /* ... */
  }
}
```

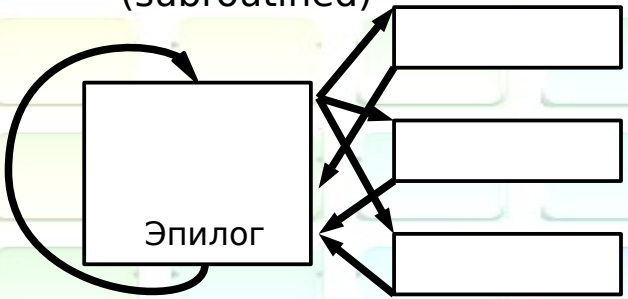


Кэширующая интерпретация (3/3)

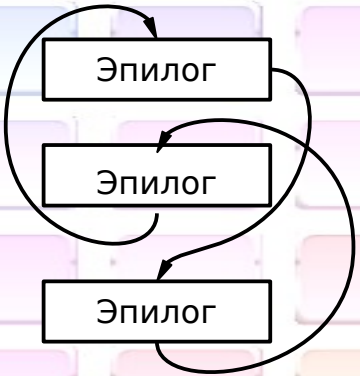
- Ёмкость кэша ограничена, следует хранить адреса, действительно часто исполняемые
- Необходимо следить за неизменностью исходного кода
- Если была запись, необходимо отбрасывать «испорченные» блоки из кэша (см. лекцию про ДТ)

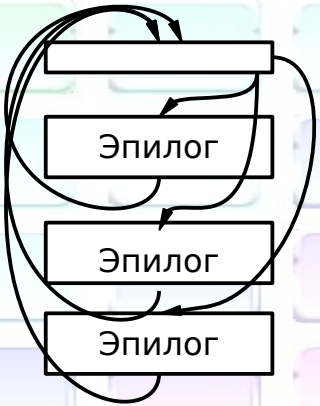
Варианты циклов интерпретации

Процедурный
(subroutined)



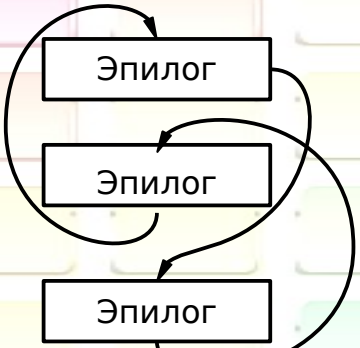
Сцепленная с метками
(threaded w/labels)





Переключаемый
(switched)

Сцепленная с хвостовыми вызовами
(threaded w/tail calls)



Что ещё можно улучшить

- Частичные вычисления функций
  - Например, если **`add rX, 1`** — частый случай, то завести для него отдельную s.r.
  - Необходима профилировка гостевого сценария
- Размещение части состояния на хозяйских регистрах (см. следующую лекцию)

Заключение

- Фазы исполнения/симуляции: F, D, E, W, A
- Decoder, disassembler, encoder
- Переключаемый (switched) И.
- Сцепленный (threaded) И.
- Кэширующий И.
- Ситуации: interrupt, trap, exception, fault

Ресурсы для дополнительного чтения

- Fredrik Larsson, Peter Magnusson, Bengt Werner. **SimGen: Development of Efficient Instruction Set Simulators**.

  `ftp://ftp.sics.se/pub/SICS-reports/Reports/SICS-R--97-03--SE.ps.Z`
- D. Mihoka , S. Shwartsman. **Virtualization Without Direct Execution or Jitting: Designing a Portable Virtual Machine Infrastructure** **`http://bochs.sourceforge.net/`**
- Yair Lifshitz, Robert Cohn, Inbal Livni, Omer Tabach, Mark Charney, Kim Hazelwood. **Zsim: A Fast Architectural Simulator for ISA Design-Space Exploration**

На следующей лекции:

- Моделирование архитектурного состояния процессора
- Моделирование доступов к памяти

Спасибо за внимание!

Все материалы курса выкладываются на сайте лаборатории:
http://iscalare.mipt.ru/material/course_materials/