

Лаборатория суперкомпьютерных технологий для биомедицины, фармакологии и малоразмерных структур

Моделирование центрального процессора с помощью интерпретации

Григорий Речистов

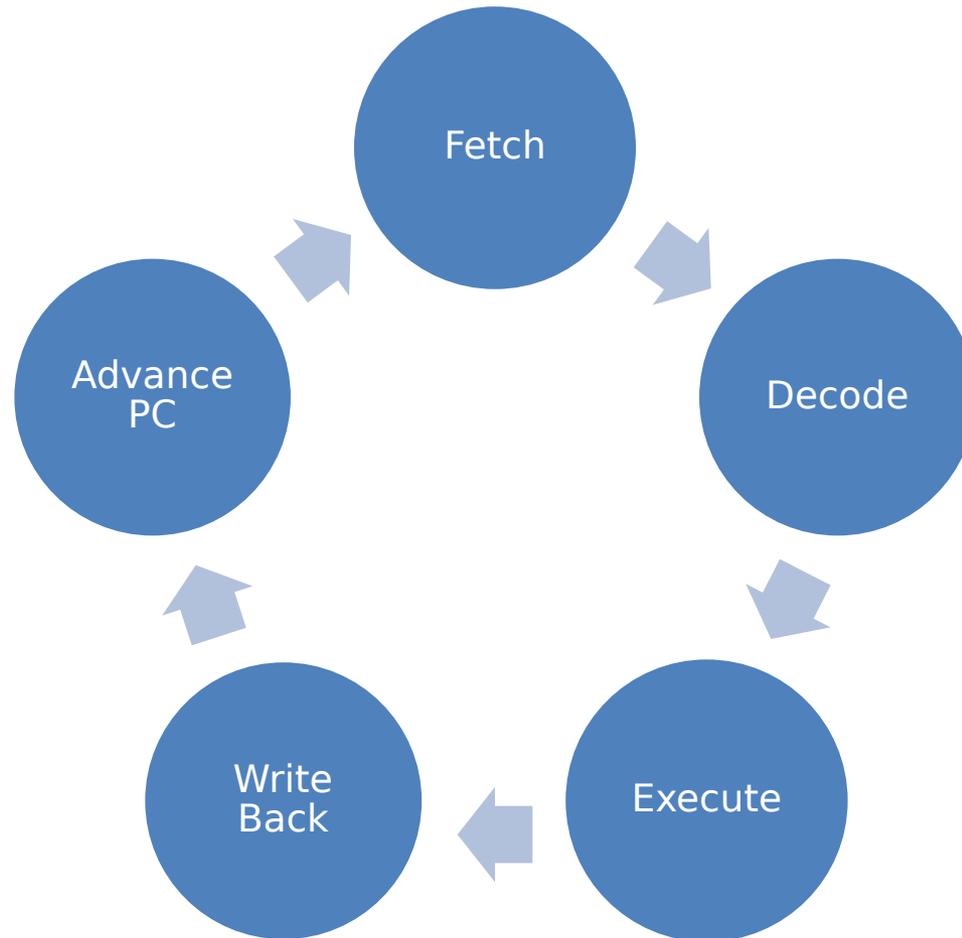
grigory.rechistov@phystech.edu

24.02.2014

На этой лекции

- Общий цикл работы процессора ↔ цикл интерпретации
- Примеры реализации
- Проблемы интерпретации
- Улучшенные схемы
- Некоторые общие вопросы симуляции

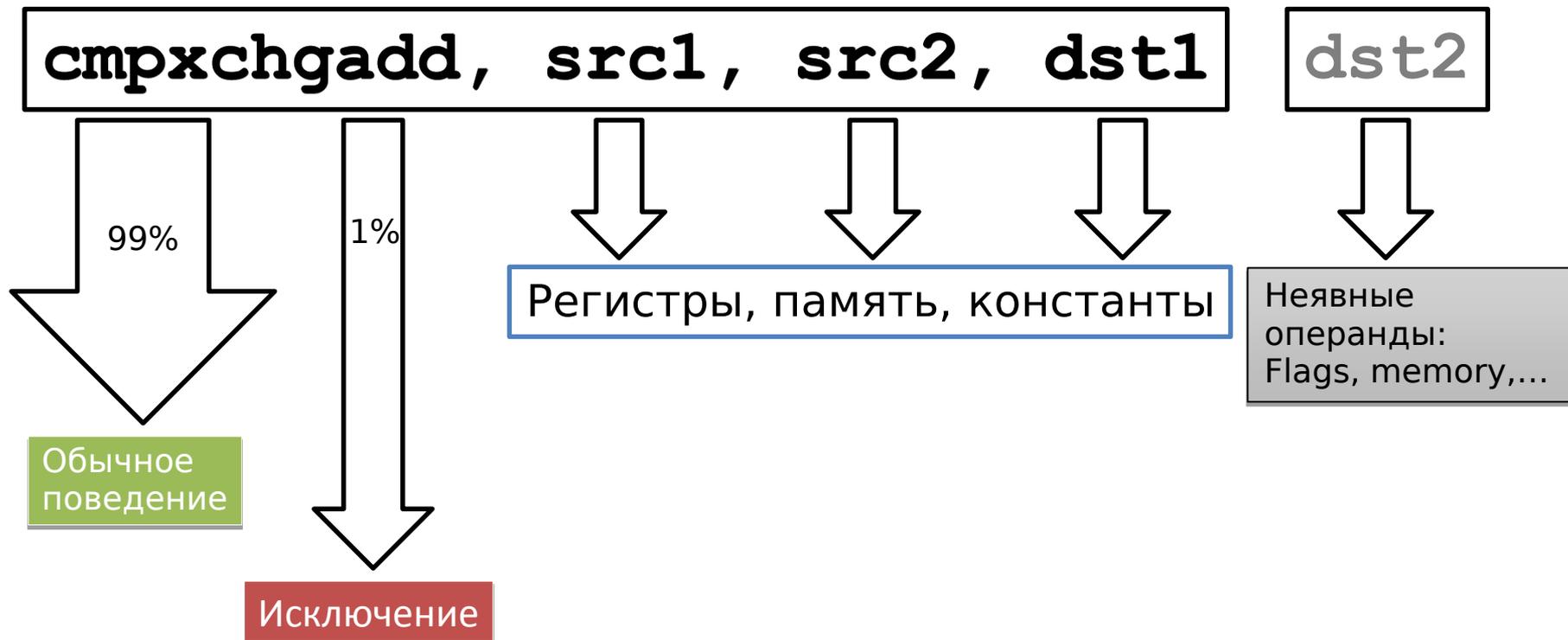
Основной цикл работы процессора



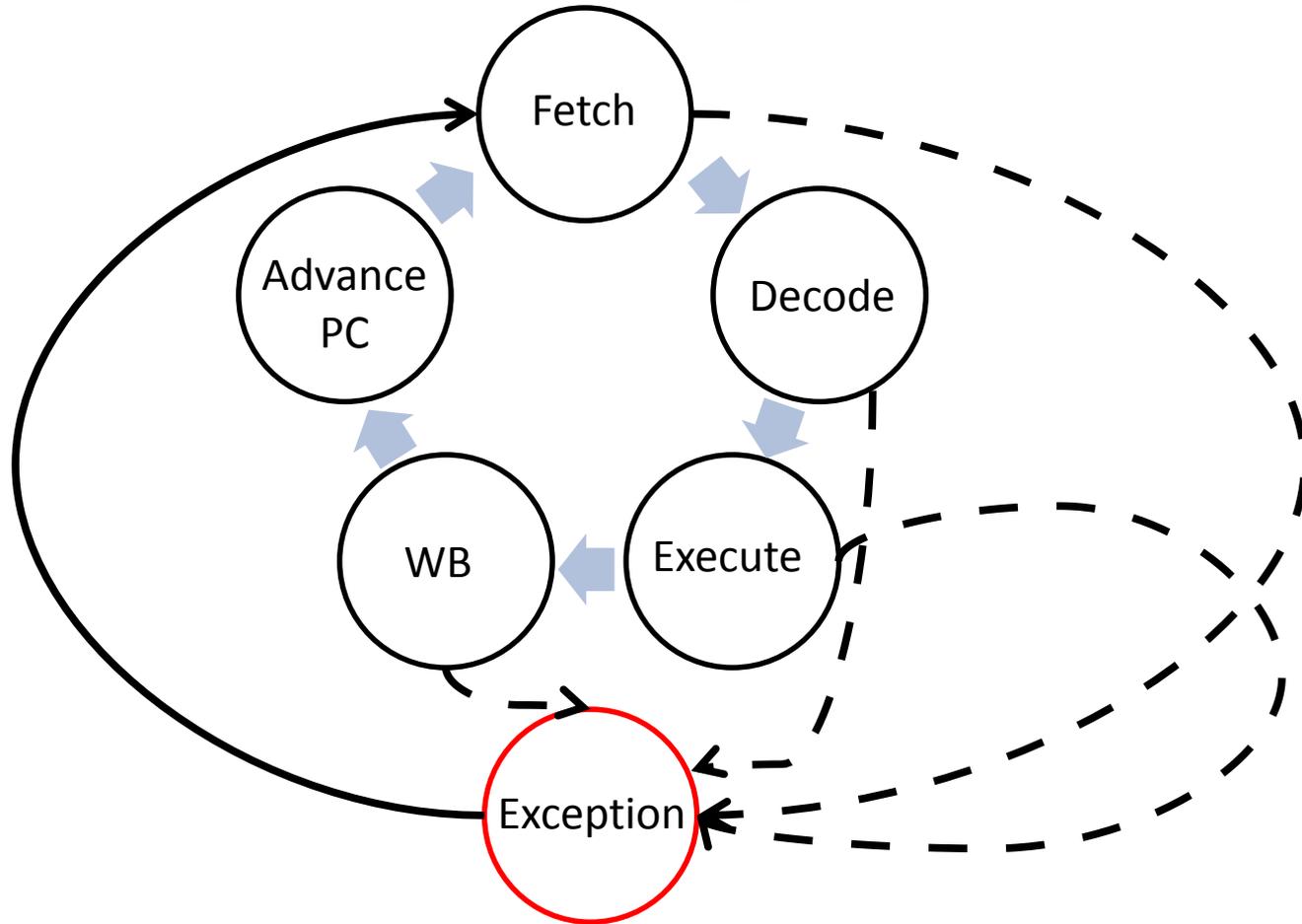
Переключаемый интерпретатор (switched)

```
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, перерыв, приостановка

Exception — синхронное исключение, без повторения текущей инструкции

Fault — синхронное, с повторением текущей инструкции

Trap — синхронные, общий случай для некоторой инструкции

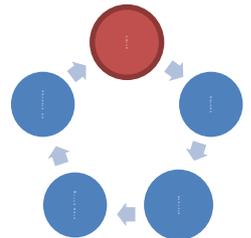
Interrupt — внешнее асинхронное прерывание

Abort — внешние асинхронное с отсутствием информации о точке возврата

Чтение инструкции из памяти

«Простое» чтение байт из памяти?

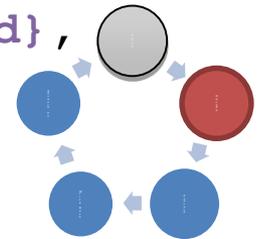
- Невыровненный (unaligned) адрес в памяти
 - Вызывает эффекты в некоторых архитектурах
- Доступ на границе двух страниц памяти
 - Разные страницы могут иметь разные характеристики



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  
}
```

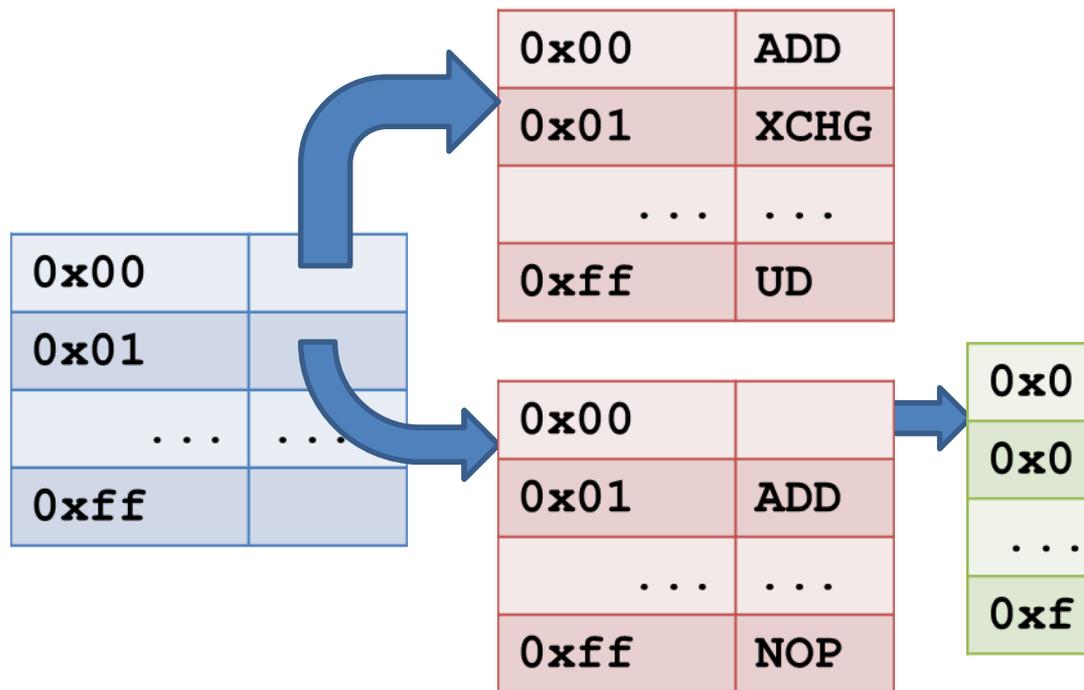


Декодирование (2/3)

Таблица опкодов

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

Таблицы префиксных кодов



Декодирование (3/3)

- Код декодера редко пишется вручную, он генерируется по описаниям

110011xxxYYY__ => sqrtpd x Y

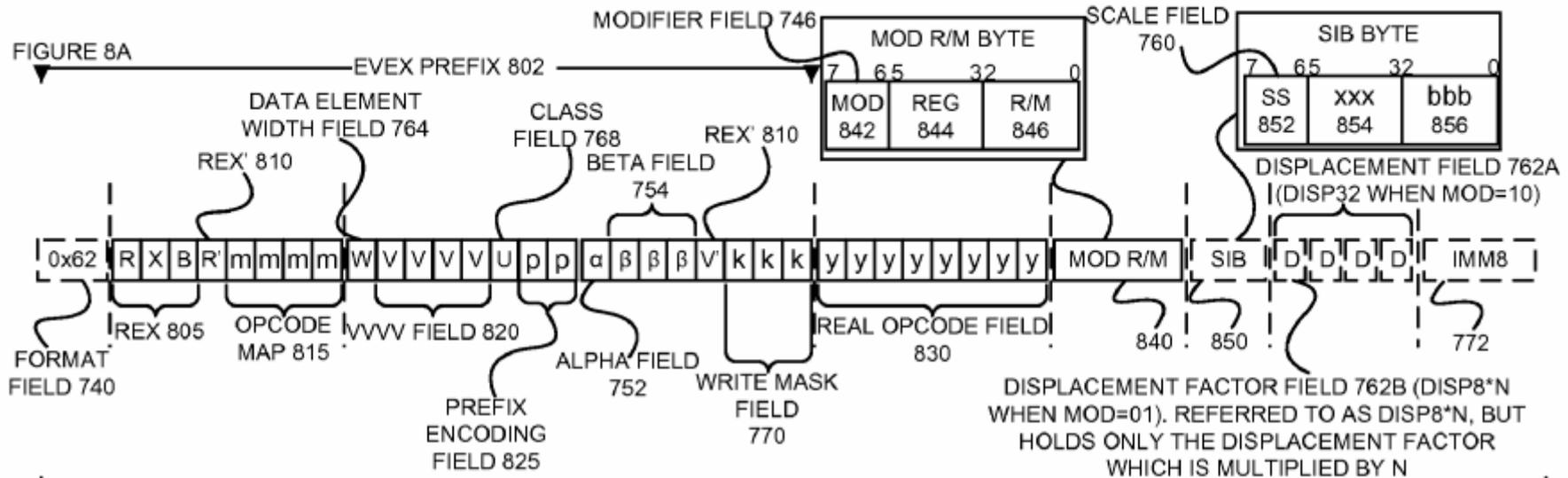
- В общем случае – классическая задача построения синтаксического анализатора
- Пример декодера – XED (X86 encoder-decoder)
- <http://www.pintool.org/docs/24110/Xed/html>

Пример: Itanium 2.3

FP Arithmetic	F1	8 - D	x	sf		f_4	f_3	f_2	f_1	qp				
Fixed Multiply Add	F2	E	x	x_2		f_4	f_3	f_2	f_1	qp				
FP Select	F3	E	x			f_4	f_3	f_2	f_1	qp				
FP Compare	F4	4	r_b	sf	r_a	p_2	f_3	f_2	t_a	p_1	qp			
FP Class	F5	5			fc_2	p_2	$fclass_{7c}$	f_2	t_a	p_1	qp			
FP Recip Approx	F6	0 - 1	q	sf	x	p_2	f_3	f_2		f_1	qp			
FP Recip Sqrt App	F7	0 - 1	q	sf	x	p_2	f_3			f_1	qp			
FP Min/Max/Pcmp	F8	0 - 1		sf	x	x_6	f_3	f_2		f_1	qp			
FP Merge/Logical	F9	0 - 1			x	x_6	f_3	f_2		f_1	qp			
Convert FP to Fixed	F10	0 - 1		sf	x	x_6		f_2		f_1	qp			
Convert Fixed to FP	F11	0			x	x_6		f_2		f_1	qp			
FP Set Controls	F12	0		sf	x	x_6	$omask_{7c}$	$amask_{7b}$			qp			
FP Clear Flags	F13	0		sf	x	x_6					qp			
FP Check Flags	F14	0	s	sf	x	x_6		imm_{20a}			qp			
Break	F15	0	i		x	x_6		imm_{20a}			qp			
Nop/Hint	F16	0	i		x	x_6	y	imm_{20a}			qp			
Break	X1	0	i	x_3		x_6		imm_{20a}			qp	imm_{41}		
Move Imm ₆₄	X2	6	i	imm_{9d}			imm_{5c}	i_c	v_c	imm_{7b}	r_1	qp	imm_{41}	
Long Branch	X3	C	i	d	wh	imm_{20b}					p	btype	qp	imm_{39}
Long Call	X4	D	i	d	wh	imm_{20b}					p	b_1	qp	imm_{39}
Nop/Hint	X5	0	i	x_3		x_6	y	imm_{20a}			qp	imm_{41}		

40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Ещё пример: IA-32 EVEX



Дизассемблирование

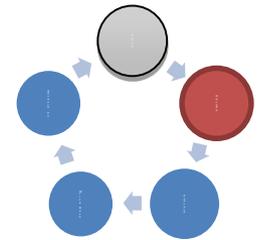
- Перевод инструкции из машинного представления в вид, понятный для человека (в мнемонику)

Закодирование (encoding)

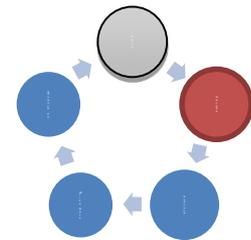
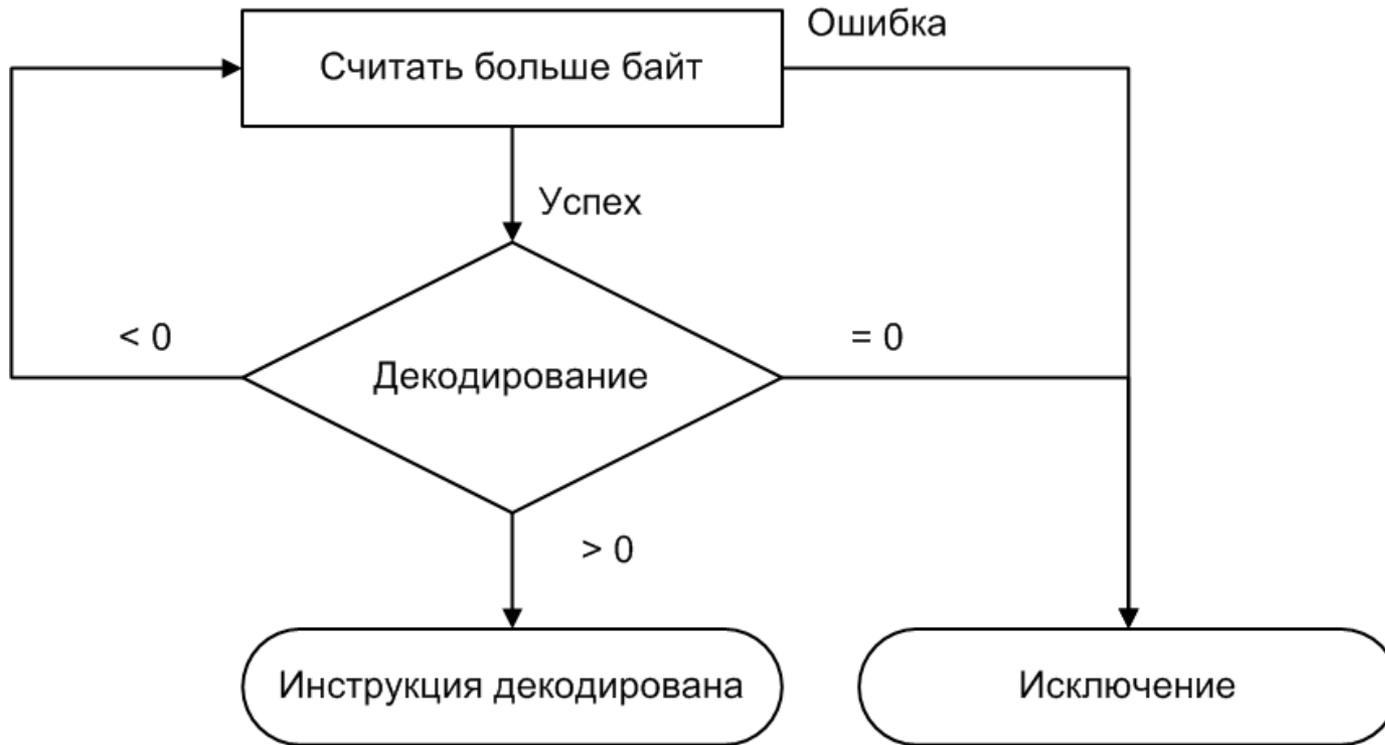
- Перевод инструкции из мнемонической записи в машинный код
- Перевод инструкции из декодированной структуры в машинный код

Декодирование: суровая реальность

- Переменная длина инструкций
 - IA-32: от 8 до 120 бит
 - Сколько байт пытаться декодировать за один раз?
- Зависимость смысла от префикса, режима работы процессора
 - Пример: `0x48`
- Полное несоответствие какому-либо здравому смыслу

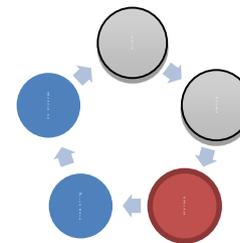


Учёт переменной длины инструкции



Исполнение

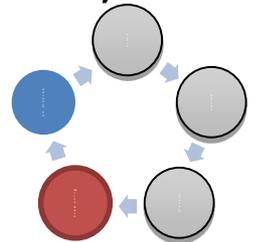
- Базовая единица — функция-эмулятор одной инструкции (service routine)
- S.r. пишутся на языке высокого уровня – переносимость между хозяйскими платформами, компиляторами
- Используются генераторы кода по описанию
 - Пример: SimGen — из одного описания совмещает генерируются декодер, дизассемблер и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



Запись результата в память

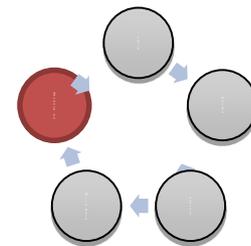
«Обычная» запись в память

- 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 (ловушки)

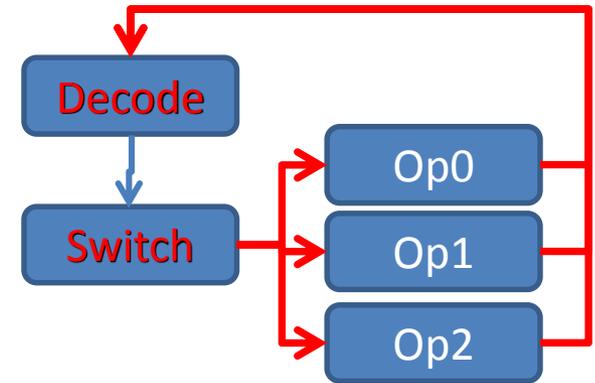


За и против интерпретации

- 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;  
    ...  
  }  
}
```



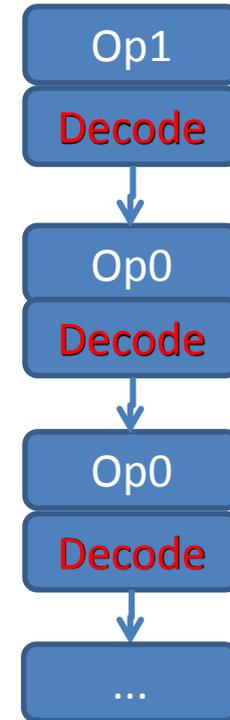
Как ускорить интерпретацию?

- Избавляемся от медленных операций
- Не делаем то, что уже сделано.
- Используем ресурсы хозяина эффективно

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];
```



Бонус-вопрос

Какой язык программирования (система программирования) высокого уровня использует сцепленное исполнение как главную технику для ускорения исполнения программ?

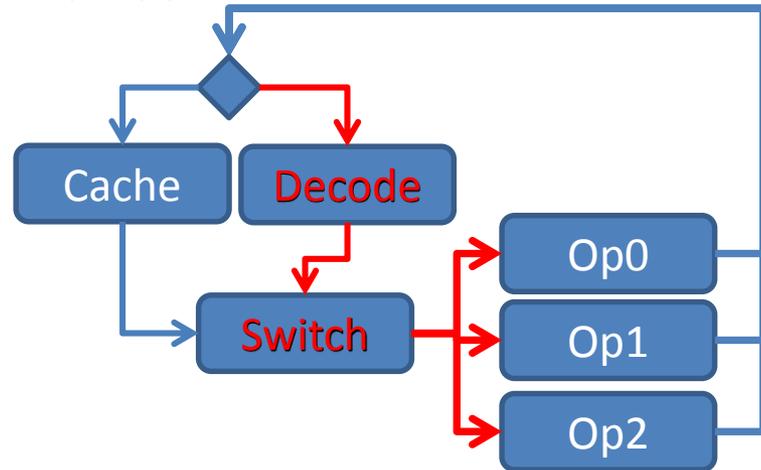
Кэширующая интерпретация (1/3)

- В большинстве случаев в код гостевого приложения неизменен
- Велика вероятность того, что инструкции с некоторыми $\$PC$ будут исполнены много раз (*задача*)
- 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
- Фаза Fetch пропускается => не совсем точная симуляция (часто это ОК)

Итоги

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

Ресурсы для дополнительного чтения 1

- D. Mihoka , S. Shwartsman. **Virtualization Without Direct Execution or Jitting: Designing a Portable Virtual Machine Infrastructure**
<http://bochs.sourceforge.net/>
- 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>
- Yair Lifshitz, Robert Cohn, Inbal Livni, Omer Tabach, Mark Charney, Kim Hazelwood. **Zsim: A Fast Architectural Simulator for ISA Design-Space Exploration**
<http://www.cs.virginia.edu/kim/docs/wish11zsim.pdf>

Ресурсы для дополнительного чтения 2

- Префиксы в системе команд IA-32
<http://habrahabr.ru/company/intel/blog/200598/>
- Программная симуляция микропроцессора. Коробка передач
<http://habrahabr.ru/company/intel/blog/202926/>

На следующей лекции:

- Моделирование архитектурного состояния ЦПУ
- Моделирование доступов в ОЗУ (функциональных)
- Преобразование адресов

Спасибо за внимание!

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

Замечание: все торговые марки и логотипы, использованные в данном материале, являются собственностью их владельцев. Представленная здесь точка зрения отражает личное мнение автора, не выступающего от лица какой-либо организации.