

# Моделирование центрального процессора с помощью интерпретации

Курс «Программное моделирование вычислительных систем»

Григорий Речистов  
[grigory.rechistov@phystech.edu](mailto:grigory.rechistov@phystech.edu)

19 февраля 2015 г.





# На прошлой лекции

Требования к симуляторам:

- точность,
- скорость,
- совместимость/расширяемость

# Вопросы

- В чём измеряется скорость симуляции?

# Вопросы

- В чём измеряется скорость симуляции?
- Как соотносятся скорости функционального и потактового симуляторов?

# Вопросы

- В чём измеряется скорость симуляции?
- Как соотносятся скорости функционального и потактового симуляторов?
- Для чего необходимо предоставлять API симулятора?

# Вопросы

- В чём измеряется скорость симуляции?
- Как соотносятся скорости функционального и потактового симуляторов?
- Для чего необходимо предоставлять API симулятора?  
Чтобы пользователи могли с ним поиграть.

# Конвейер процессора





# Переключаемый интерпретатор (switched)

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

# Чтение инструкции из памяти

```
data = mem[pc];
```

# Чтение инструкции из памяти

```
data = mem[pc];  
Скопее,  
data = read_mem(pc);
```

# Порядок байт при доступах

- Порядок от младшего к старшему (*англ.* little-endian);
- Порядок от старшего к младшему (*англ.* big-endian);
- Смешанный порядок (*англ.* middle-endian).

Представление	$D4 + C3 \times 0x100 + B2 \times 0x10000 + A1 \times 0x1000000$
Little-endian	D4, C3, B2, A1
Big-endian	A1, B2, C3, D4

# Бит, байт, слово

- Бит — ?

# Бит, байт, слово

- Бит — ?
- Байт

# Бит, байт, слово

- Бит — ?
- Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации

# Бит, байт, слово

- Бит — ?
- Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации
- Октет



# Бит, байт, слово

- Бит — ?
- Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации
- Октет — восемь бит

# Бит, байт, слово

- Бит — ?
- Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации
- Октет — восемь бит
- Машинное слово

# Бит, байт, слово

- Бит — ?
- Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации
- Октет — восемь бит
- Машинное слово — максимальный объём информации, который ЦПУ может обработать одновременно

# Бит, байт, слово

- Бит — ?
- Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации
- Октет — восемь бит
- Машинное слово — максимальный объём информации, который ЦПУ может обработать одновременно

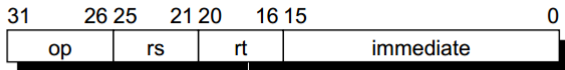
Intel: word — 16 бит, dword — 32 бит, qword — 64 бит.

# Декодирование

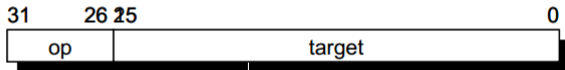
Перевод данных об инструкции из машинного представления во внутреннее (высокоуровневое), удобное для последующего анализа

# Пример: MIPS

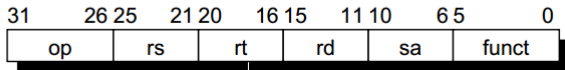
## I-Type (Immediate)



## J-Type (Jump)



## R-Type (Register)



## Пример: код

```
#define BITS(v, s, e) (v >> s) & (( 1 << (e-s+1)) - 1)
typedef struct decode { /* ... */} decode_t;
static inline int32_t sign_extend(uint32_t v, int width)
    { /* ... */ };
decode_t decode(uint32_t raw) {
    uint32_t op = BITS(raw, 26, 31);
    uint32_t rs = BITS(raw, 21, 25);
    uint32_t rt = BITS(raw, 16, 20);
    int32_t  imm = sign_extend(BITS(raw, 0, 15));
    int32_t  tgt = sign_extend(BITS(raw, 0, 25));
    /* ... */
    uint32_t funct = BITS(raw, 0, 5);
    return decode_t{op, rs, rt, imm, tgt, funct};
}
```

# Более сложный пример: Intel IA-64 2.3

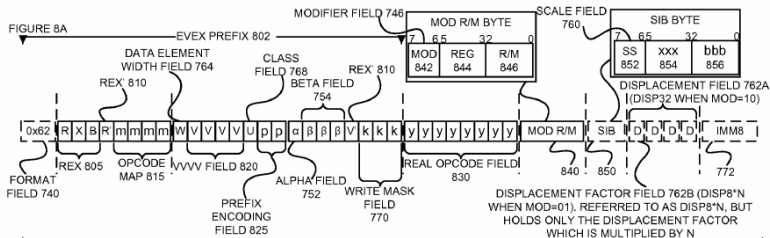
FP Arithmetic	F1	8 - D	x	sf			f <sub>4</sub>		f <sub>3</sub>		f <sub>2</sub>		f <sub>1</sub>		qp
Fixed Multiply Add	F2	E	x	x <sub>2</sub>			f <sub>4</sub>		f <sub>3</sub>		f <sub>2</sub>		f <sub>1</sub>		qp
FP Select	F3	E	x				f <sub>4</sub>		f <sub>3</sub>		f <sub>2</sub>		f <sub>1</sub>		qp
FP Compare	F4	4	r <sub>b</sub>	sf	r <sub>a</sub>		p <sub>2</sub>		f <sub>3</sub>		f <sub>2</sub>		t <sub>a</sub>	p <sub>1</sub>	qp
FP Class	F5	5			fc <sub>2</sub>		p <sub>2</sub>		fclass <sub>7c</sub>		f <sub>2</sub>		t <sub>a</sub>	p <sub>1</sub>	qp
FP Recip Approx	F6	0 - 1	q	sf	x		p <sub>2</sub>		f <sub>3</sub>		f <sub>2</sub>		f <sub>1</sub>		qp
FP Recip Sqrt App	F7	0 - 1	q	sf	x		p <sub>2</sub>		f <sub>3</sub>				f <sub>1</sub>		qp
FP Min/Max/Pcmp	F8	0 - 1		sf	x		x <sub>6</sub>		f <sub>3</sub>		f <sub>2</sub>		f <sub>1</sub>		qp
FP Merge/Logical	F9	0 - 1			x		x <sub>6</sub>		f <sub>3</sub>		f <sub>2</sub>		f <sub>1</sub>		qp
Convert FP to Fixed	F10	0 - 1			sf	x	x <sub>6</sub>				f <sub>2</sub>		f <sub>1</sub>		qp
Convert Fixed to FP	F11	0				x	x <sub>6</sub>				f <sub>2</sub>		f <sub>1</sub>		qp
FP Set Controls	F12	0			sf	x	x <sub>6</sub>		omask <sub>7c</sub>		amask <sub>7b</sub>				qp
FP Clear Flags	F13	0			sf	x	x <sub>6</sub>								qp
FP Check Flags	F14	0	s		sf	x	x <sub>6</sub>								qp
Break	F15	0	i			x	x <sub>6</sub>								qp
Nop/Hint	F16	0	i			x	x <sub>6</sub>	y							qp
Break	X1	0		i	x <sub>3</sub>		x <sub>6</sub>								qp
Move Imm <sub>64</sub>	X2	6		i			imm <sub>9d</sub>		imm <sub>5c</sub>	i <sub>c</sub>	v <sub>c</sub>	imm <sub>7b</sub>		r <sub>1</sub>	qp
Long Branch	X3	C		i	d	wh							p	btype	qp
Long Call	X4	D		i	d	wh							p	b <sub>1</sub>	qp
Nop/Hint	X5	0		i	x <sub>3</sub>		x <sub>6</sub>	y							qp

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

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

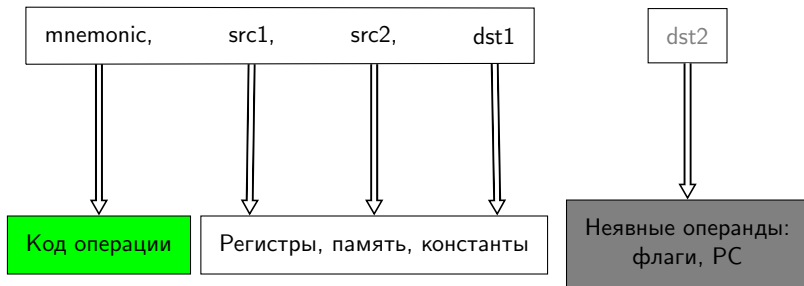


# Пример посложнее: Intel IA-32



J.C.S. Adrian et al. Systems, Apparatuses, and Methods for Blending Two Source Operands into a Single Destination Using a Writemask. US Patent Application Publication. № 2012/0254588 A1

# Что извлекать из машинного кода инструкции



На выходе декодера:

- Успех, неуспех, недостаточно данных
- Для успеха: длина инструкции

# Декодирование

Код декодера редко пишется вручную, чаще он генерируется по описанию.

A5 YX 0Z 00  $\Rightarrow$  MOD RX, RY, RZ

В общем случае: классическая задача построения синтаксического анализатора.

На практике: специализированные инструменты и языки.

# 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

# Дизассемблирование

Дизассемблирование — перевод инструкций из машинного представление в понятный человеку вид (мнемонику).

# Дизассемблирование

Дизассемблирование — перевод инструкций из машинного представление в понятный человеку вид (мнемонику).

(За)кодирование (encoding) — перевод инструкций из мнемонической записи в машинный код.

# Исполнение

- Базовая единица — функция-эмулятор одной инструкции (service routine).
- Service routine пишутся на языке высокого уровня — переносимость кода между хозяйскими платформами, компиляторами.

## Пример: ADD

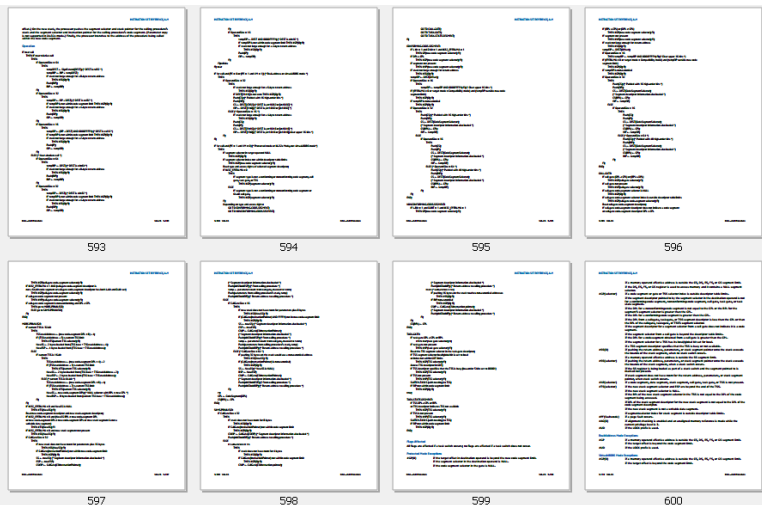
```
void add32_rr(cpu_t *cpu, int src1, int src2, int dst) {  
    cpu->regs[dst] = cpu->regs[src1] + cpu->regs[src2];  
}
```



## Пример: ADD

```
void add32_rr(cpu_t *cpu, int src1, int src2, int dst) {  
    cpu->regs[dst] = cpu->regs[src1] + cpu->regs[src2];  
  
    cpu->z_flag = cpu->regs[dst] == 0;  
    cpu->n_flag = cpu->regs[dst] & (1 << 31);  
    cpu->o_flag = cpu->regs[dst] <  
        MAX(cpu->regs[src1], cpu->regs[src2]);  
    cpu->c_flag = calc_c_flag(cpu->regs[src1],  
        cpu->regs[src2]);  
}
```

# Пример посложнее: IA-32 CALL



## Запись результата в память

```
write_mem(cpu, dst_addr, data);
```

# Запись результата в память

```
write_mem(cpu, dst_addr, data);
```

- Невыровненный адрес
- Граница страниц
- Запись в память «только для чтения»
- Исключения для всего или части записи/чтения

# Уточнённый цикл работы процессора



# Классификация

- 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 — внешнее асинхронное с отсутствием информации о точке возврата

# Продвижение PC

- Для большинства команд: увеличение счетчика на длину обработанной инструкции.  
`cpu->pc += instr_length;`

# Продвижение PC

- Для большинства команд: увеличение счетчика на длину обработанной инструкции.  
`cpu->pc += instr_length;`  
Бывают исключения: REP MOVSB в IA-32.



# Продвижение PC

- Для большинства команд: увеличение счетчика на длину обработанной инструкции.  
`cpu->pc += instr_length;`  
Бывают исключения: REP MOVSB в IA-32.
- Явное изменение PC — команды управления исполнением:
  - (Без)условные (не)прямой прыжок/переход
  - Вызов/возврат из процедуры
  - Системный вызов

# Преимущества и недостатки интерпретации

- Пишется на языках высокого уровня: код переносим
- Простая структура: надёжность, расширяемость, переиспользование

# Преимущества и недостатки интерпретации

- Пишется на языках высокого уровня: код переносим
- Простая структура: надёжность, расширяемость, переиспользование
- (Очень) низкая скорость работы

# Куда тратится время?

TODO Напиши **меня** картинка

# Куда тратится время?

TODO Напиши **меня** картинка

- Неэффективное использование аппаратуры
- Ненужные вызовы декодера

# 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:

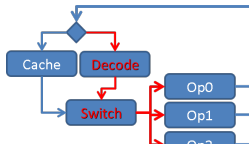
```
/* simulate instr2 */; PC++;  
next_opcode = decode(fetch(PC));  
goto func_ptr[next_opcode];
```

# Кэширующая интерпретация

В большинстве случаев в код гостевого приложения неизменен  
Велика вероятность того, что инструкции с некоторыми РС  
будут исполнены много раз (задача) Зачем каждый раз  
декодировать их? Заводим таблицу `addr =>`  
`decoded_instruction`

# Кэширующая интерпретация

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








# Кэширующая интерпретация

- Ёмкость кэша ограничена
- Необходимо следить за неизменностью исходного кода

# 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

# Литература I

-  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>

# На следующей лекции

## Моделирование архитектурного состояния

# Спасибо за внимание!

*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.