

Моделирование центрального процессора с помощью интерпретации

Курс «Программное моделирование вычислительных систем»

Григорий Речистов
grigory.rechistov@phystech.edu

2 марта 2015 г.

- 1 Конвейер процессора
- 2 Fetch
- 3 Decode
- 4 Execute
- 5 Write Back
- 6 Исключения
- 7 Advance PC
- 8 Улучшенные схемы

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?

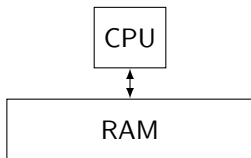
Вопросы

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

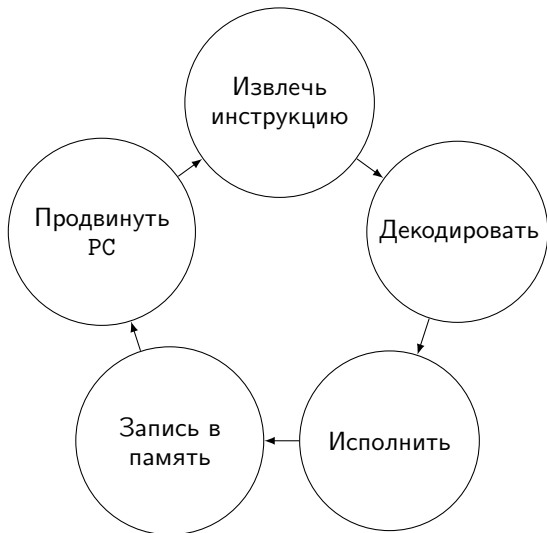
Вопросы

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

Моделируемая система



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];
```

Скорее,

```
data = read_mem(pc);
```

Чтение инструкции из памяти

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

Скорее,

```
data = read_mem(pc);
```

И не забыть про преобразование адресов:

```
paddr = v2p(pc); // pc - vaddr
```

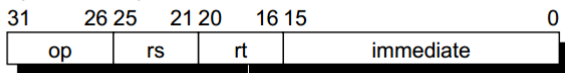
```
data = mem[paddr];
```

Декодирование

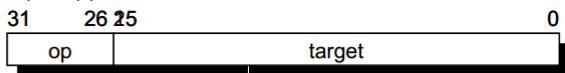
Перевод данных об инструкции из машинного представления (последовательность байт) во внутреннее (высокоуровневое), удобное для последующего использования

Пример: MIPS

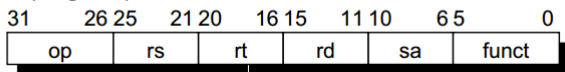
I-Type (Immediate)



J-Type (Jump)



R-Type (Register)



MIPS Technologies Inc. MIPS32 4K™ Processor Core Family Software User's Manual - 2002

Пример: код 1

```
#define BITS(v, s, e) (v >> s) & (( 1 << (e-s+1)) - 1)

typedef struct decode {
    uint32_t op;
    uint32_t rs;
    uint32_t rt;
    int32_t  imm;
    int32_t  tgt;
    /* ... */
} decode_t;

static inline
int32_t sign_extend(uint32_t v, int width)
    { /* ... */};
```


Пример: код 2

```
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, /* ... */, funct};
}
```

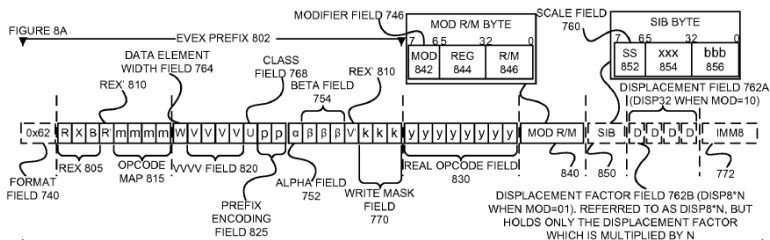
Более сложный пример: Intel IA-64 2.3

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		f_2		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
Move Imm ₆₄	X2	6	i			imm_{9d}		imm_{5c}	i_c	v_c	imm_{7b}	r_1		qp
Long Branch	X3	C	i	d	wh			imm_{20b}				p	btype	qp
Long Call	X4	D	i	d	wh			imm_{20b}				p	b_1	qp
Nop/Hint	X5	0	i		x_3		x_6	y			imm_{20a}			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

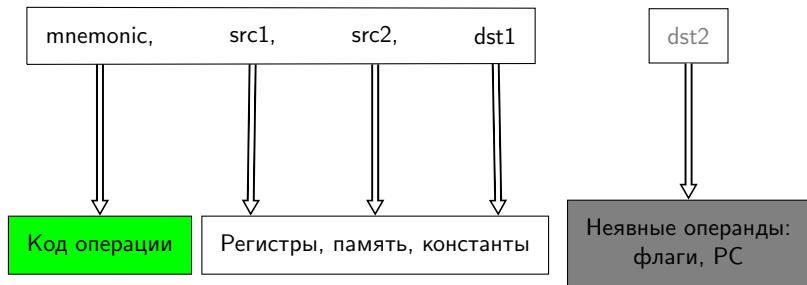
Intel Corporation. Intel® Itanium® Architecture Software Developer's Manual, p 3:296

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
- Для успеха: эмулирующая процедура

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: 0x40–0x4f в IA-32/Intel 64/AMD64
- Полное несоответствие какому-либо здравому смыслу

Дизассемблирование

- Дизассемблирование — перевод инструкций из машинного представление в понятный человеку вид (мнемонику)
- (За)кодирование (encoding) — перевод инструкций из мнемонической записи в машинный код

Дизассемблирование

- Дизассемблирование — перевод инструкций из машинного представление в понятный человеку вид (мнемонику)
- (За)кодирование (encoding) — перевод инструкций из мнемонической записи в машинный код

Вопрос: однозначны ли операции: декодирования, дизассемблирования, кодирования?

Исполнение

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

Симулируемое состояние

```
typedef struct {  
    uint32_t regs[16];  
    bool z_flag;  
    bool n_flag;  
    bool o_flag;  
    bool c_flag;  
    uint32_t pc;  
} cpu_t;
```

Пример: ADD reg reg reg

```
void add32_rrr(cpu_t *cpu, int src1, int src2, int dst) {  
    cpu->regs[dst] = cpu->regs[src1] + cpu->regs[src2];  
}
```

Пример: ADD reg reg reg

```
void add32_rrr(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]);  
}
```


Запись результата в память

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

Запись результата в память

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

- Невыровненный адрес
- Граница страниц
- Запись в память «только для чтения»
- Исключения для всей 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 — синхронное, без повторения, намеренно вызванное
- 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?

```
start: interruption = false;
while (!interruption) {
    raw_code = fetch(PC);
    (opcode, operands) = decode(raw_code); // <-- здесь
    switch (opcode) { // <-- здесь
    case opcode1:
        func1(operands); PC++; break;
    case opcode2:
        func2(operands); PC++; break;
    /*...*/
    }
}
handle_interruption();
goto start;
```

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];
```

<http://stackoverflow.com/questions/11227809/>

[why-is-processing-a-sorted-array-faster-than-an-unsorted-array](#)

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 их декодировать?
- Заводим таблицу соответствия «адрес инструкции → декодированный результат»

Кэширующая интерпретация

```
while (!interruption) {
    if (operation = cache[PC]); // короткий путь
    else { // не в кэше, длинный путь
        operation = decode(fetch(PC));
        cache[PC] = operation; // на будущее
    }
    switch (operation) {
        /* ... */
    }
}
```




Кэширующая интерпретация

- Ёмкость любого кэша ограничена, старые данные надо выбрасывать
- Необходимо следить за неизменностью исходного кода, иначе сохранённое соответствие будет неверным

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>

Литература II



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

Спасибо за внимание!

Слайды и материалы курса доступны по адресу
<http://is.gd/ivuboc>

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.