

# Моделирование центрального процессора с помощью интерпретации

Курс «Программное моделирование вычислительных систем»

Коньчев Валерий  
[valerakonychev@gmail.com](mailto:valerakonychev@gmail.com)

September 25, 2015



# На этой лекции

- 1 Цикл работы процессора
- 2 Fetch
- 3 Decode
- 4 Execute
- 5 Write Back
- 6 Advance PC
- 7 Практика

# Цикл работы 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

«Простое» чтение байт из памяти?

# Чтение инструкции из памяти

«Простое» чтение байт из памяти?

- Невыровненный (*англ.* *unaligned*) адрес в памяти. Вызывает эффекты в некоторых архитектурах.

# Чтение инструкции из памяти

«Простое» чтение байт из памяти?

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

# Порядок байт при доступах



## 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).

## 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).

Пример	0xA1B2C3D4
Little-endian	D4, C3, B2, A1
Big-endian	A1, B2, C3, D4
Middle-endian	B2, A1, D4, C3

# Бит, байт, слово

Бит

# Бит, байт, слово

Бит — наименьшая единица информации.

# Бит, байт, слово

Бит — наименьшая единица информации.

Байт

# Бит, байт, слово

Бит — наименьшая единица информации.

Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации.

# Бит, байт, слово

Бит — наименьшая единица информации.

Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации.

Октет — восемь бит.

# Бит, байт, слово

Бит — наименьшая единица информации.

Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации.

Октет — восемь бит.

Машинное слово



# Бит, байт, слово

Бит — наименьшая единица информации.

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 — наименьшая единица информации.

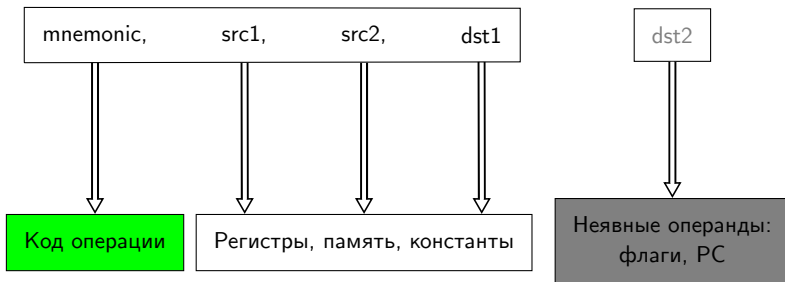
Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации.

Октет — восемь бит.

Машинное слово — максимальный объём информации, который ЦПУ может обработать одновременно.

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

# Анатомия инструкции



# Декодирование

Задача декодирования — перевод данных об инструкции из машинного представление во внутреннее (высокоуровневое) удобное для последующего анализа.

# Декодирование

Задача декодирования — перевод данных об инструкции из машинного представление во внутреннее (высокоуровневое) удобное для последующего анализа.

Вход: 0x40 0x05 0xab 0x12

Результат:

```
instruction {  
    opcode = ADDI, num_operands = 2,  
    dst = {type = OP_REG, reg = R5},  
    src = {type = OP_IMM, val = 0x12ab},  
    disasm = "addi r5, 0x12ab",  
    addr = 0x1234  
}
```



# Декодирование

Код декодера редко пишется вручную, он генерируется по описанию:

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

# Декодирование

Код декодера редко пишется вручную, он генерируется по описанию:

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

В общем случае классическая задача построения синтаксического анализатора.

# Декодирование

Код декодера редко пишется вручную, он генерируется по описанию:

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

В общем случае классическая задача построения синтаксического анализатора.

Пример декодера — XED (x86 encoder-decoder)

<https://software.intel.com/sites/landingpage/pintool/docs/61206/Xed/html/>.





# Дизассемблирование

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

# Дизассемблирование

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

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

# Исполнение

Базовая единица — функция-эмулятор одной инструкции (service routine).

s.r. пишутся на языке высокого уровня — переносимость кода между хозяйскими платформами, компиляторами.

Используются генераторы кода.

Пример: SimGen — из одного описания генерируются декодер, дизассемблер и s.r.

# Запись результата в память

«Обычная» запись в память:

# Запись результата в память

«Обычная» запись в память:

- Невыровненные адрес,
- Граница страниц,
- Попытка изменить регион памяти доступный только для чтения,
- Часть результата может быть записана, а потом случится исключение.

# Продвижение \$PC

- Для большинства команд увеличение счетчика на длину обработанной инструкции.  
Исключение: REP MOVSB.

# Продвижение \$PC

- Для большинства команд увеличение счетчика на длину обработанной инструкции.  
Исключение: REP MOVSB.
- Явное изменение \$PC — команды управления исполнением:
  - (Un)conditional (In)direct Jump/Branch,
  - Call/Return (subroutine).

# Структура кода

Файл `modules/chip16/chip16.h`

`struct chip16_t;` — состояние процессора.



# Структура кода

Файл `modules/chip16/chip16.h`

`struct chip16_t`; — состояние процессора.

Файл `modules/chip16/chip16.c`

Функции:

- `chip16_string_decode` — дизассемблер,
- `chip16_execute` — декодирование и исполнение,
- `cr_register_attributes` — регистрация атрибутов.

# Тестирование

```
test/chip16/unit-tests/nop/s-nop.py
```

```
# This test checks NOP instruction
```

```
import stest
```

```
cli.run_command("run-python-file _%/targets/chip16/machine.py" %  
                conf.sim.workspace)
```

```
def test_nop_availability(cpu):
```

```
    paddr = 0
```

```
    cpu.pc = paddr
```

```
    # NOP
```

```
    simics.SIM_write_phys_memory(cpu, paddr, 0, 4)
```

```
    SIM_continue(1)
```

```
    stest.expect_equal(cpu.pc, paddr + 4)
```

```
    print "NOP: _success"
```

```
test_nop_availability(conf.chip0)
```

```
Запуск: > bin/test-runner test/chip16/unit-tests/nop
```



# Инструкции для реализации

- 1 SUBI RX, HHLLL;
- 2 AND RX, RY;
- 3 OR RX, RY, RZ;
- 4 XOR RX, RY, RZ;
- 5 MUL RX, RYL.

# 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.