

# Моделирование архитектурного состояния

## Курс «Программное моделирование вычислительных систем»

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

16 марта 2015 г.

- 1 Регистровый файл
- 2 Ленивое вычисление состояния
- 3 Большие массивы
- 4 Регистры, поля, банки
- 5 Прерывания
- 6 Endianness

## На прошлой лекции

Интерпретаторы — простой метод создания моделей процессоров

# Вопросы

- Чем декодирование отличается от дизассемблирования?

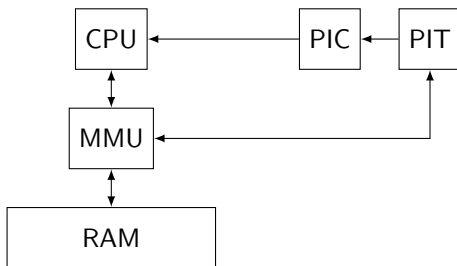
# Вопросы

- Чем декодирование отличается от дизассемблирования?
- Чем прерывание отличается от исключения?

# Вопросы

- Чем декодирование отличается от дизассемблирования?
- Чем прерывание отличается от исключения?
- Может ли сцепленный интерпретатор быть кэширующим?

# Платформа



# Регистровый файл

Register					r31	r30
Register	r29	r28	r27	r26	r25	r24
Register	r23	r22	r21	r20	r19	r18
Register	r17	r16	r15	r14	r13	r12
Register	r11	r10	r9 LR	r8	r7	r6
Register	r5	r4	r3	r2	r1	r0

**Table 4-3. General-Purpose Registers**

Grp #	Reg #	Reg Name	USER MODE	SUPV MODE	Description
0	0	VR	–	R	Version register
0	1	UPR	–	R	Unit Present register
0	2	CPUCFGR	–	R	CPU Configuration register
0	3	DMMUCFGR	–	R	Data MMU Configuration register
0	4	IMMUCFGR	–	R	Instruction MMU Configuration register
0	5	DCCFGR	–	R	Data Cache Configuration register
0	6	ICCFGR	–	R	Instruction Cache Configuration register

OpenCores. OpenRISC 1000 Architecture Manual rev 1.0



# C-структура

```
#include <stdint.h>
#define NO_GPRS 32
#define NO_SPRS 1024
#define REG_VR 0
#define REG_UPR 1
/* ... */
typedef struct cpu {
    uint32_t pc;
    uint32_t gprs[NO_GPRS];
    uint32_t sprs[NO_SPRS];
} cpu_t;
```

# Хозяйские регистры

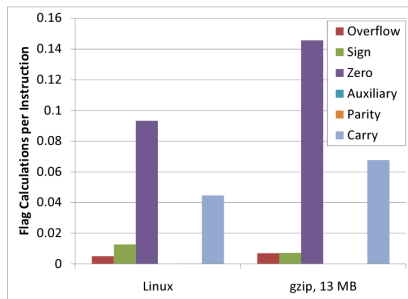
- Доступ к структуре в памяти медленный
- Доступ к хозяйским регистрам быстрее
- Разместим гостевые регистры на хозяйских!
- `register cpu_t* cpu __asm__("rbp");`

# Хозяйские регистры

- Все гостевые регистры могут не поместиться
- Есть смысл размещать только самые часто используемые
- Потеря переносимости кода
- Специализация регистров: PC, SP, FP[0-7]
- Мы мешаем работе компилятора: давление на регистры

# Ленивое вычисление состояния

- Вообще не вычислять состояние, пока оно не понадобится!
- Пример: IA-32 EFLAGS
- $flags = f(a, b, op)$  для последней инструкции, влияющей на флаги



Yair Lifshitz et al. Zsim: A Fast Architectural Simulator for ISA Design-Space Exploration

## Доступ к ОЗУ и большим массивам

Просто выделить на куче!

```
uint8_t *buf = (uint8_t *)calloc(BUF_SIZE, 1);
```

```
/* read */
```

```
uint8_t val = buf[offset];
```

```
/* write */
```

```
buf[offset] = val;
```

## Доступ к ОЗУ и большим массивам

Просто выделить на куче!

```
uint8_t *buf = (uint8_t *)calloc(BUF_SIZE, 1);
```

```
/* read */
```

```
uint8_t val = buf[offset];
```

```
/* write */
```

```
buf[offset] = val;
```

Что же делать, если,  $\text{BUF\_SIZE} \sim 2^{53}$ , а хозяйской памяти в наличии  $2^{32}$ ?

## Ленивое выделение памяти

Для непрерывного диапазона адресов хранилище выделяется по мере необходимости кусками фиксированного размера (страницами).

```
page = addr & PAGE_MASK;
hptr = lookup_hptr(page);
if (!hptr)
    hptr = allocate_hptr(page);
assert(hptr);
haddr = hptr + (addr & PAGE_OFFSET);
```

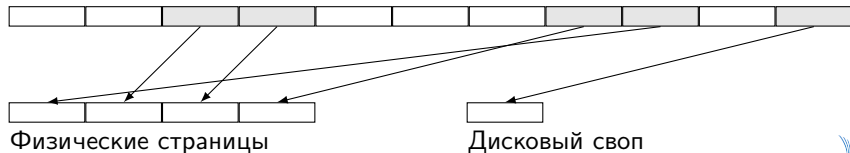
При недостатке хозяйской памяти «старые» страницы выгружаются на диск.

## Звучит знакомо?

- Это же виртуальная память!
- POSIX-системы предоставляют механизм `mmap()`

```
void *mmap(void *addr,  
           size_t len, int prot, int flags,  
           int fildes, off_t off);
```

Выделенный блок





## Два способа взаимодействия с регистрами устройств

PIO — programmable I/O, выделенные инструкции для общения с периферией

```
IN EAX, DX
```

```
OUT DX, EAX
```

## Два способа взаимодействия с регистрами устройств

PIO — programmable I/O, выделенные инструкции для общения с периферией

```
IN EAX, DX  
OUT DX, EAX
```

MMIO — memory mapped I/O, унифицированный подход к доступу к ОЗУ и устройствам

```
MOV [mem], reg  
MOV reg, [mem]
```

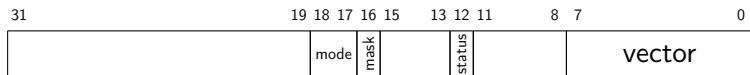
# Операции над регистрами

- read, write, fetch, prefetch
- Регистры-хранилище и регистры с побочными эффектами
- Примеры регистров с side-effects: time stamp (RW), command (W), status (R), version (RO)
- inquiry — «призрачное» обращение (без эффектов)
- reset

# Операции над регистрами

```
template <class rtype> class IRegister {  
    virtual Exception Read(rtype& retval) = 0;  
    virtual Exception Fetch(rtype& retval) = 0;  
    virtual Exception Prefetch(rtype& retval) = 0;  
    virtual Exception Write(const rtype& newval) = 0;  
    virtual bool InquiryRead(rtype& retval) = 0;  
    virtual bool InquiryWrite(const rtype& newval) = 0;  
    virtual void Reset() = 0;  
}
```

# Битовые поля

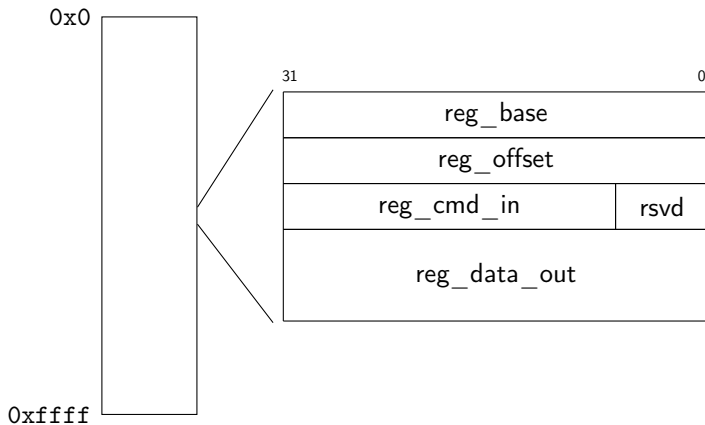


APIC LVT Timer. Intel® 64 and IA-32 Architectures Software Developer's Manual

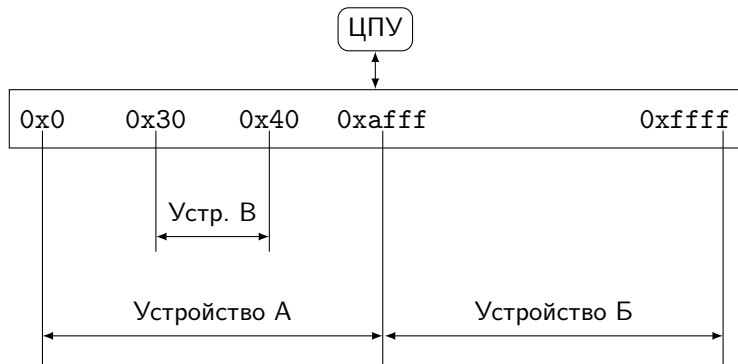
- Настоящие базовые единицы спецификаций и моделирования
- Могут иметь совершенно различные свойства внутри регистра

# Банк регистров

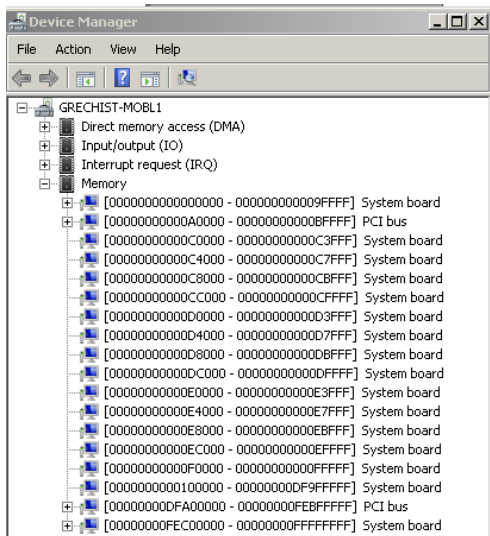
Группа регистров устройства, находящиеся в одной области



# Карты памяти

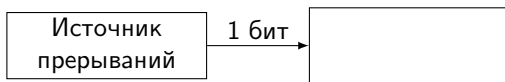


# Пример: devmgmt.msc





# Прерывания

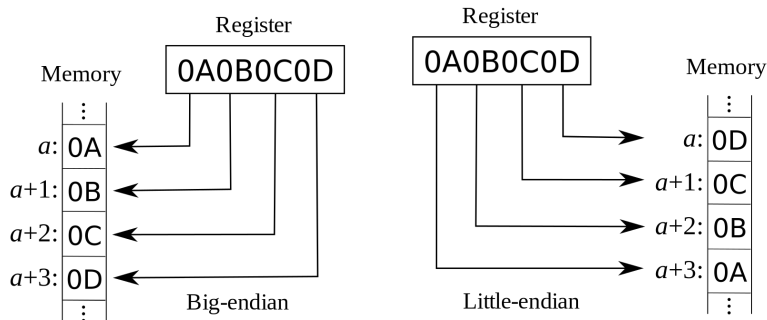


- Как моделировать отдельное прерывание — очень просто:  
`take_interrupt(dev_t *dev);`
- Когда моделировать — вопрос сложнее, тема отдельной лекции

# Преобразование адресов

- v2p
- p2h
- $v2p + p2h = v2h$

# Порядок байт при доступах



# Бит, байт, слово

- Бит — кол-во информации, уменьшающее неопределённость в два раза

# Бит, байт, слово

- Бит — кол-во информации, уменьшающее неопределённость в два раза
- Байт

# Бит, байт, слово

- Бит — кол-во информации, уменьшающее неопределённость в два раза
- Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации

# Бит, байт, слово

- Бит — кол-во информации, уменьшающее неопределённость в два раза
- Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации
- Октет

# Бит, байт, слово

- Бит — кол-во информации, уменьшающее неопределённость в два раза
- Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации
- Октет — восемь бит



# Бит, байт, слово

- Бит — кол-во информации, уменьшающее неопределённость в два раза
- Байт — минимальная адресуемая (в данной архитектуре) единица хранения информации
- Октет — восемь бит
- Машинное слово

# Бит, байт, слово




- Бит — кол-во информации, уменьшающее неопределённость в два раза
- 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 — кол-во информации, уменьшающее неопределённость в два раза
- 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.

# Литература I

-  Stanislav Shwartsman, Darek Mihoka. How Bochs Works Under the Hood. 2nd edition. <http://bochs.sourceforge.net/HowtheBochsworksunderthehood2ndedition.pdf>
-  M. Domeika, M. Loenko, P. Ozhdikhin, E. Brevnov. Bi-Endian Compiler: A Robust and High Performance Approach for Migrating Byte Order Sensitive Applications. <http://world-comp.org/p2011/ESA2902.pdf>
-  Leonid Baraz [et al.] IA-32 Execution Layer: a Two-Phase Dynamic Translator Designed to Support IA-32 Applications on Itanium®-Based Systems. <http://www.microarch.org/micro36/html/pdf/goldenberg-IA32ExecutionLayer.pdf>

## На следующей лекции

- Делаем симулятор, работающий быстрее, чем интерпретатор
- Двоичная трансляция

# 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.