


# Архитектура ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

## Семинар № 9



Работа со строкам



# План семинарского занятия

---

## Цель и задачи

Изучение команд и методов, обеспечивающих обработку строк символов в кодировки ASCII (иные кодировки различны в разных ОС или требуют дополнительных алгоритмических решений)

## Основные вопросы

1. Особенности обработки символов в процессоре RISC-V.
2. Организация работы с файлами в RARS.
3. Примеры обработки текстов

# Работа со строками

ASCII



# Всё, что нужно знать о строках

---

1. Строки — последовательности ненулевых байтов, заканчивающихся символом с кодом 0 (однобайтным нулём) согласно ASCII
2. Ввод заканчивается символом перевода строки - ASCII-код «10»  
Но размер буфера под строку должен быть на 2 больше - '\n' и '0'
3. Некорректный ввод (пустая строка) приводит к исключению.  
Но в RARS всё хуже (пока): ввод пустой строки может привести к аварийному останову самого RARS.
4. Мы не можем вводить строки произвольной длины — разве что в случае, когда в обход операционной системе размещаем вводимую строку в конце Кучи. Но и тогда нужно было бы проверять, не упёрлись ли мы в стек...

# Подсчёт количества пробелов

```
1 .eqv      SIZE 100          # размер буфера
2 .data
3 str:      .space SIZE       # буфер
4 .text
5          la      a0 str      # считаем строку в буфер
6          li      a1 SIZE
7          li      a7 8
8          ecall
9          mv      t0 zero     # счётчик пробелов
10         li      t2 0x20     # пробел
11 loop:    lb      t1 (a0)     # очередной символ
12         beqz    t1 fin      # нулевой — конец строки
13         bne     t1 t2 nosp   # не пробел — обойдём счётчик
14         addi    t0 t0 1      # пробел — увеличим счётчик
15 nosp:    addi    a0 a0 1     # следующий символ
16         b       loop
17 fin:     mv      a0 t0       # выведем счётчик
18         li      a7 1
19         ecall
20         li      a7 10
21         ecall
```

# Статическая память

---

- Выделяется секций `.data`
- Начиная с `0x10010000` резервируются области, которые затем доступны с помощью меток
- Области могут быть инициализированы начальными значениям (`.word`, `.asciz` и т. п.), а могут остаться без изменения (`.space`).
- Добавление новых директив резервирования памяти в секцию `.data` приведёт к тому, что соответствующие ячейки окажутся после уже отведённых.
- Можно хранить данные только в течение определённого времени – (локальные переменные) временные данные хранятся в стеке.

# Заказ памяти у системы

---

В других случаях память необходимо выделять на всё время работы программы, но размер их заранее неизвестен.

Один раз это можно сделать, воспользовавшись в качестве базового адреса **началом кучи** — 0x10040000. Однако в дальнейшем придётся где-то **хранить вершину кучи**, чтобы следующая область динамически выделяемой на куче памяти не пересекалась с предыдущей.

**В RARS этим занимается операционная система.**  
**Системный вызов 9**, которому в `a0` передаётся объём заказываемой памяти, отведёт на куче соответствующую область и вернёт в `a0` её адрес.

# Подпрограмма, которая заказывает память

```
1 newmem: li      a7 9                # Заказем память (объём уже в a0)
2         ecall
3         addi     sp sp -4
4         sw       a0 (sp)           # Сохраним адрес
5         li       a7 34            # Выведем адрес (a0 уже задан)
6         ecall     # как 16-ричное число
7         li       a0 10            # Выведем перевод строки
8         li       a7 11
9         ecall
10        lw       a0 (sp)           # Вспомним адрес
11        addi     sp sp 4
12        ret
```

```
0x10040000
0x10040100
0x10040204
```

```
1 .globl main
2 main:  li       a0 0x100
3        jal      newmem
4        li       a0 0x101
5        jal      newmem
6        li       a0 0x100
7        jal      newmem
```



# Освобождают память? Нет, не слышал, я программирую на C#

---

- Необходимо иметь возможность заказывать фрагменты памяти произвольного размера
- Необходимо иметь возможность освободить произвольные заказанные ранее фрагменты памяти в произвольное время
- Следовательно, относительно каждого фрагмента необходимо постоянно хранить метainформацию — размер и местоположение.
- Надо решить проблему фрагментации: если мы заказали 2048 фрагментов размером в килобайт (2 Мб суммарно), а затем освободили каждый второй, у нас не образовалось 1 Мб свободной памяти! Вместо этого у нас образовалось 1024 килобайтных фрагмента, никакие два из которых не идут в памяти подряд

# Системные вызовы

Кто именно исполняет esall?



# ecall — Environment Call

---

Кто именно исполняет ecall?

Например, в RARS ecall — это вообще функции Java.

**ecall** — это обращение к ядру «операционной системы», однако что именно называется операционной системой — такой же код, только в режиме супервизора, обращение к системе виртуализации в обход ядра, непосредственно к аппаратуре и т. п.

В целом можно сказать, что программа запрашивает какие-то данные из «окружения», в котором она выполняется.

# Аппаратная поддержка

---

**Задачи операционной системы:**

унификация, разделение и учёт ресурсов компьютера.

**Мы знаем**, что окружение, в котором запускается программа, умеет выполнять какие-то функции

**Мы не имеем возможности** самостоятельно эти функции реализовать

# Вариант: «СИСТЕМНЫЙ ВЫЗОВ»

---

Универсальная пользовательская инструкция «обратиться к ОС» — «системный вызов» (syscall)

1. Фактически это — вызов подпрограммы, в котором вместо адреса используется заданный в API/ABI номер системного вызова.
2. Прозрачный (невидимый пользователю) механизм выполнения
3. Защита памяти ядра
4. Поддержка минимум двух потоков вычисления (пользовательская программа и ядро ОС)
5. Разделение режимов работы процессора на режим ядра (полный доступ к ресурсам) и режим пользователя (ограниченный доступ и/или доступ к виртуализованным посредникам)

# Вариант с обобщением «функция окружения», или `ecall`:

---

От полностью программного (выполняется машинный код из области ядра, фактически `syscall`) обращение к супервизору мимо ядра ОС до полностью аппаратного (нужную операцию выполняет компьютер)

И в случае `syscall`, и в случае `ecall` эта функция обычно реализована аппаратно, отдельной инструкцией.

# Функции окружения RARS

---

Будем называть **ecall** по-старинке: **системный вызов** механизм программного обращения к ядру операционной системы.

На архитектурах, имеющих аппаратную поддержку режима ядра, реализован отдельной инструкцией.

Конвенции по вызову **ecall** в RISC-V

1. В регистр **a7** помещается номер системной функции (service number)
2. В регистры **a\*** или **fa\*** помещаются параметры системного вызова
3. Инструкция **ecall** передаёт управление операционной системе
4. Возврат из системного вызова — по аналогии с возвратом из подпрограммы, на следующую после **ecall** инструкцию
5. Возвращаемые значения (если есть) помещаются в **a\*** или в **fa0**

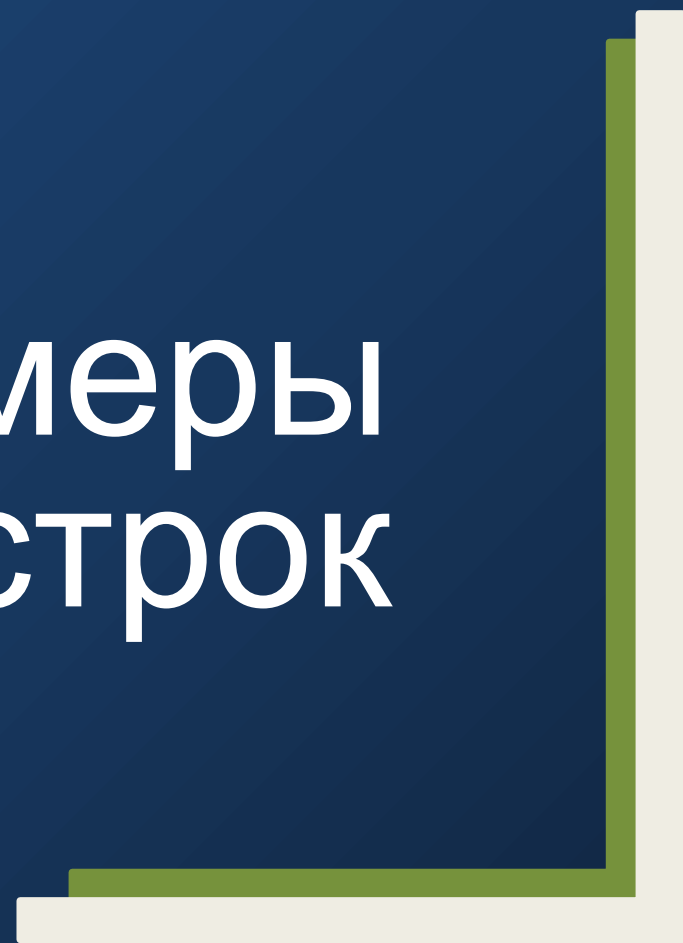
# Например

---

- вызовы 40-44 — это работа с генератором случайных чисел (важно: не с аппаратной случайностью),
- вызовы 50-60 (кроме 57) — замена вызовам ввода-вывода, использующая диалоговые окна Java
- вызовы 17, 57, 62-64 и (внезапно) 1024 — работа с файлами на вашем компьютере
- вызов 93 — останов программы и всего RARS-а (как 10-й), но с возвратом в вашу операционную систему кода завершения
- 31 и 33 — проигрывание ноты на аудиокарте вашего компьютера



# Примеры обработки строк



# Моделирование функции strlen

```
6  int strlen(char* ascii_string) {
7      int i = 0;
8      for(i = 0; ascii_string[i] != '\0'; ++i);
9      return i;
10 }
```

```
12 int main()
13 {
14     char buf[BUF_SIZE];
15     printf("Input string: ");
16     scanf("%s", buf);
17     int count = strlen(buf);
18     printf("String length = %d\n", count);
19     count = strlen("Hello");
20     printf("String length = %d\n", count);
```

*Пример: 03-strlen*

```

9  .text
10 strlen:
11     li      t0 0      # Счетчик
12 loop:
13     lb      t1 (a0)    # Загрузка символа для сравнения
14     beqz    t1 end
15     addi    t0 t0 1    # Счетчик символов увеличивается
16     addi    a0 a0 1    # Берется следующий символ
17     b       loop
18 end:
19     mv      a0 t0
20     ret
21 fatal:
22     li      a0 -1
23     ret

```

```

24 .globl main
25 main:
26     # Ввод строки в буфер
27     la      a0 buf
28     li      a1 BUF_SIZE
29     li      a7 8
30     ecall
31     # Тестовый вывод строки
32     la      a0 buf
33     li      a7 4
34     ecall
35     # Вычисление длины для вводимой строки
36     la      a0 buf
37     jal     strlen
38     # Вывод счетчика
39     li      a7 1
40     ecall
41     # Перевод строки
42     li      a0 '\n'
43     li      a7 11
44     ecall

```

# Вычисление длины строки, ограниченной нулем или числом анализируемых символов

Основная идея подобного ограничения является обычно в **дополнительной проверке буфера на переполнение**, которое может возникать по разным причинам. Например, при наложении данных друг на друга. Это возможно в небезопасных языках программирования, к которым относятся Си и Ассемблеры.

```
int strn_len(char* asciiz_string, int max_size) {  
    int i = 0;  
    for(i = 0; asciiz_string[i] != '\0'; ++i) {  
        if(i >= max_size) return -1;  
    }  
    return i;  
}
```

*Пример: 04-strnlen*

# Сравнение на равенство двух строк

Основная идея заключается в обработке и сопоставлении СИМВОЛЬНЫХ ДАННЫХ.

```
int str_cmp(char* string1, char* string2) {  
    int i = 0;  
    for(i = 0; (string1[i] != '\0') || (string2[i] != '\0'); ++i) {  
        if(string1[i] != string2[i]) break;  
    }  
    return string1[i] - string2[i];  
}
```

```
result = str_cmp("Hello", "Hi");  
printf("Result = %d\n", result);
```

*Пример: 05-strcmp*

# Примеры подпрограмм и макроосов обработки строк символов

---

Рассмотрим варианты объединения кода в подобие библиотек подпрограмм и макроопределений.

Для этого подпрограммы из предыдущих примеров вынесены в отдельные файлы (можно было их все вынести в один общий файл)

*Пример: 06-string-macro*

# Домашнее задание

---

Написать подпрограмму, осуществляющую копирование строки символов аналогично функции **strncpy** языка программирования C. Протестировать функцию на различных комбинациях данных. Ознакомиться с функцией можно в системе справки по библиотеке языка C, которая имеется в различных источниках информации. Исходные данные для тестирования задавать как при вводе с консоли, так и с использованием строк символов в разрабатываемой программе (по аналогии с программами, рассмотренными на семинаре). Подпрограмму вынести в отдельный файл.

## Опционально до +2 баллов

Дополнительно к подпрограмме разработать соответствующий макрос, расширив тем самым макробиблиотеку строк символов.