



Архитектура ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Семинары №3



Методы адресации.
Ветвления и переходы.
Системные вызовы.



План семинарского занятия

Цель и задачи

- Методы адресации данных, используемые в ассемблере RISC-V.
- Моделирование на ассемблере различных операторов управления.
- Имитация в симуляторе системных вызовов. Аналогии системных вызовов в операционных системах.
- Примеры простых целочисленных алгоритмов.

Основные вопросы

1. Обзор методов адресации.
2. Команды и псевдокоманды условных и безусловных переходов.
3. Имитация операторов управления высокого уровня.

Методы адресации используемые в ассемблере RISC-V

реализованные в эмуляторе RARS



Непосредственная адресация

AUIPC rd, immediate $\# \text{rd} \leftarrow \text{PC} + \text{immediate}[31:12] \ll 12$

Add Upper Immediate to PC (добавить константу к старшим битам PC).

```
.data
var:      .word    0xbadface

.text
        li        t5 0x10010000
        la        t6 var
```

Сохраняем **старшие 20 бит смещения** до некоторой метки в верхние 20 бит регистра rd.

Смещение JALR может быть шириной **12 бит**, и вместе мы получаем $(20 + 12 = 32)$, что **даёт нам 32-битный адрес**

0x00400000	0x10010f37	lui x30,0x00010010	5	li	t5 0x10010000
0x00400004	0x000f0f13	addi x30,x30,0x00000000			
0x00400008	0x0fc10f97	auipc x31,0x0000fc10	6	la	t6 var
0x0040000c	0xff8f8f93	addi x31,x31,0xffffffff8			

Косвенная адресация

.data

```
        .word 0x1223344  
var:    .word 0xdeadbeef  
addr:   .word var
```

.text

```
lw t1 var  
lw t2 addr  
lw t3 (t2)  
lw t4 4(t2)  
lw t5 -4(t2)
```

Косвенная адресация

```
1 .data
2         .word    0x1223344
3 var:    .word    0xdeadbeef
4 addr:   .word    var
5 .text
6         lw       t1 var
7         lw       t2 addr
8         lw       t3 (t2)
9         lw       t4 4(t2)
10        lw       t5 -4(t2)
```

0x00400000	0x0fc10317	auipc x6,0x0000fc10	6	lw	t1 var
0x00400004	0x00432303	lw x6,0x00000004(x6)			
0x00400008	0x0fc10397	auipc x7,0x0000fc10	7	lw	t2 addr
0x0040000c	0x0003a383	lw x7,0x00000000(x7)			
0x00400010	0x0003ae03	lw x28,0x00000000(x7)	8	lw	t3 (t2)
0x00400014	0x0043ae83	lw x29,0x00000004(x7)	9	lw	t4 4(t2)
0x00400018	0xffc3af03	lw x30,0xffffffffc(x7)	10	lw	t5 -4(t2)

Загрузка из памяти и переходы

SW, SH и SB

Инструкция LW загружает 32-битное значение из памяти в rd. LH загружает 16-битное значение из памяти, затем расширяет знак до 32-бит перед сохранением в rd. Инструкции SW, SH и SB сохраняют в памяти 32-битные, 16-битные и 8-битные значения из младших битов регистра rs2.

beq, bne, blt, bge, btlu, bgeu

Инструкции перехода сравнивают два регистра. BEQ и BNE переходят к переходу, если регистры rs1 и rs2 равны или не равны соответственно. BLT и BLTU выбирают ветвь, если rs1 меньше, чем rs2, используя знаковое и беззнаковое сравнение соответственно. BGE и BGEU переходят к переходу, если rs1 больше или равно rs2, используя знаковое и беззнаковое сравнение соответственно

.space

.align

Работа с одномерными массивами

```
#include <stdio.h>

int array[16];

int main()
{
    fill:
    for(int i = 0; i < 16; ++i) {
        array[i] = i+1;
    }
    printf("-----\n");
    out:
    for(int i = 0; i < 16; ++i) {
        printf("%d\n", array[i]);
    }
    return 0;
}
```

Если нужно обработать *массив* данных, косвенная адресация — единственный способ.

Массив — это адрес в памяти и длина

В ассемблере предпочтительнее манипулировать адресным пространством нужной величины, а не умножать каждый раз на величину слова.

Основной режим работы - использование косвенной адресации с индексацией относительно начала массива.

Как выделить память под массив:

- выделить пространство требуемого размера;
- при наличии заранее определенных значений элементов можно их перечислить;
- при неизвестном числе элементов можно зарезервировать некоторый *большой кусок памяти*, а количество элементов задавать числом, меньшим выделенного размера, которое (как и в программе на Си) может служить ограничителем цикла при формировании массива;
- можно выделить память под массив на куче после получения числа элементов в массиве.

Пример 1

Массив слов расписывается последовательными значениями:

```
.data
array: .space 64
arrend:

.text
    la t0 array
    la t1 arrend
    li t2 1
loop:
    sw t2 (t0)
    addi t2 t2 1
    addi t0 t0 4
    bltu t0 t1 loop

    li a7 10 # Останов
    ecall
```

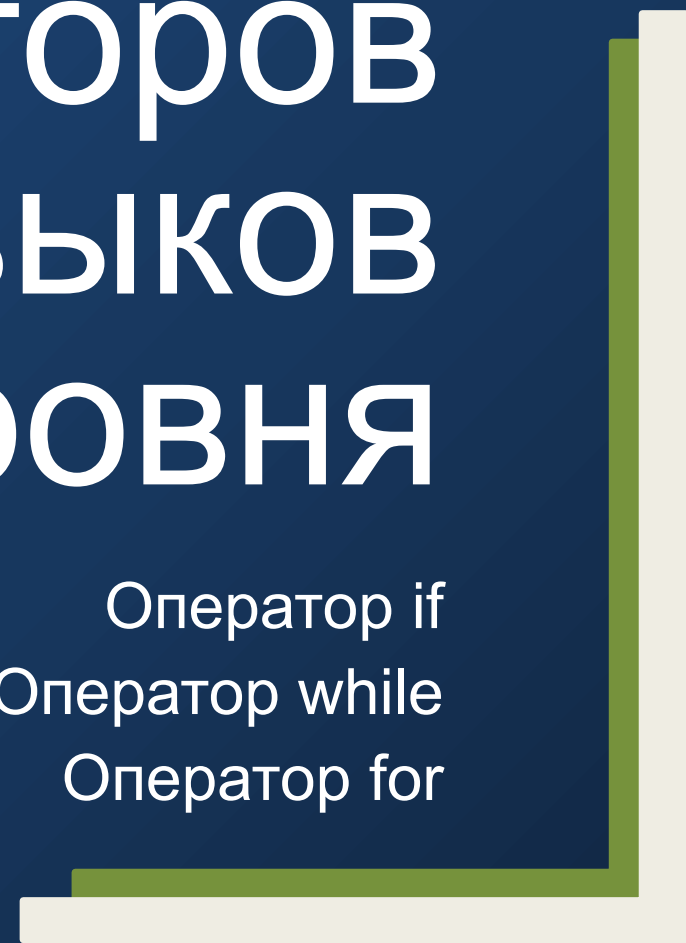
«адресная арифметика» — на каждом проходе цикла для доступа к следующему элементу массива к адресу надо прибавлять размер элемента

Адреса можно сравнивать на $>$ / $<$, но **сравнение должно быть беззнаковое**: мало ли, в какую область памяти будет загружена программа (в RARS, где адрес загрузки фиксирован)

Косвенная адресация — единственный способ обработки массива. Массив — это адрес в памяти и длина (количество элементов, умноженное на размер одного элемента). В примере массив слов расписывается последовательными значениями.

Имитация операторов управления языков высокого уровня

Оператор if
Оператор while
Оператор for



Имитация оператора if

```
# Example:
#
# if (t0 == 0) {
#     t1 = 1;
# } else if (t0 < 0) {
#     t1 = 2;
# } else if (t0 > 10) {
#     t1 = 3;
# } else {
#     t1 = 4;
# }
```

```
main:
    li    a7, 5
    ecall
    mv    t0, a0
if_0:
    bnez  t0, if_less_0
    li    t1, 1
    j     end_if
if_less_0:
    bgez  t0, if_greater_10
    li    t1, 2
    j     end_if
if_greater_10:
    li    t3, 10
    ble   t0, t3, else
    li    t1, 3
    j     end_if
else:
    li    t1, 4
end_if:
    li    a7, 1
    mv    a0, t1
    ecall
```

Имитация оператора while

Example:

#

while((t0 = read_int()) != 0) {

print_int(t0)

print_char('\n')

}

while:

li a7, 5

ecall

mv t0, a0

beqz a0, end_while

li a7, 1

ecall

li a7, 11

li a0, '\n'

ecall

j while

end_while:

Имитация оператора for

Example:

#

for (t0 = 0; t0 < t1; ++t0) {

print_int(t0)

print_char('\n')

}

for:

li a7, 5

ecall

mv t1, a0

mv t0, zero

next:

bge t0, t1, end_for

mv a0, t0

li a7, 1

ecall

li a7, 11

li a0, '\n'

ecall

addi t0, t0, 1

j next

end_for:

Пример. Алгоритм Евклида

```
#  
# Calculates the greatest common divisor of  
# two values using the Euclidean algorithm.  
#  
# function gcd(a, b)  
#     while a ≠ b  
#         if a > b  
#             a := a - b  
#         else  
#             b := b - a  
#     return a
```

Программа на ассемблере размещена в LMS

Пример. Числа Фибоначчи

```
#
# Example that calculates the Fibonacci sequence.
#
main:
    mv    t0, zero
    li    t1, 1

    li    a7, 5
    ecall

    mv    t3, a0
fib:
    beqz  t3, finish
    add   t2, t1, t0
    mv    t0, t1
    mv    t1, t2
    addi  t3, t3, -1
    j     fib
finish:
    li    a7, 1
    mv    a0, t0
    ecall
```


Домашнее задание

Оценка до 8 баллов

Разработать на ассемблере RARS программу, осуществляющую целочисленное деление для 32-разрядных целых чисел со знаком, используя операции вычитания, ветвления и циклы. Исходные делимое и делитель вводятся с клавиатуры в десятичной системе счисления. Полученные в результате деления частное и остаток необходимо вывести на консоль симулятора. Остаток от деления вычисляется по правилам, используемых при выполнении операции вычисления остатка (%) в языках программирования C/C++. Необходимо осуществлять проверку входных данных на корректность. При делении учитывать знаки операндов и результатов, а также возможность ошибок при делении на ноль. Сформировать тестовое покрытие, охватывающее проверку различных возможных комбинаций делимого и делителя. Осуществить прогон программы для этих комбинаций. В отчете привести примеры скриншотов консоли, демонстрирующие все возможные комбинации тестового покрытия.

Рекомендации. Предварительно данную программу можно отработать на языках более высокого уровня (рекомендуется использовать C/C++), чтобы посмотреть какие результаты порождают имитируемые операции.

Опционально до +2 баллов

Дополнительно к написанной программе сформировать программу, которая перебирает все рассмотренные варианты тестового покрытия, автоматизируя тем самым процесс тестирования.