МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Организация ЭВМ и систем»
Тема: Знакомство с рабочей средой эмулятора Ripes для работы с процессором RISC-V. Базовый ISA, система команд, состав регистров. Разработка и выполнение простой программы на ассемблере RISC-V.

Студентка гр. 2382 _____ Ульянова Е.А.

Преподаватель _____ Морозов С.М.

Санкт-Петербург
2023

Цель работы.

Знакомство с рабочей средой эмулятора Ripes для работы с процессором RISC-V. Базовый ISA, система команд, состав регистров. Разработка и выполнение простой программы на ассемблере RISC-V.

Задание.

1. Разработайте процедуру на ассемблере, которая для целочисленных 32-битных входных переменных x, y, z и констант a, b, c вычисляет выражение:

$$R = f(x, y, z, a, b, c)$$

В выражениях используются следующие константы:

Константа	Значение
a	[Сумма цифр студ. билета]
b	[Количество букв в фамилии]
c	[Количество букв в полном имени]

Вариант 16

R = ((z ^ (-b)) ^ (x & c)) & (y << a)

Константы:

a = 17 (№ студ. билета 238220)

b = 8 (Ульянова)

c = 9 (Екатерина)

Формула для вычисления:

R = ((z ^ (-8)) ^ (x & 9)) & (y << 17)

2. Напишите программу, которая для двух наборов исходных данных x, y, z выполняет вычисление заданного выражения с помощью разработанной процедуры, сохраняет в регистрах и выводит на экран результаты вычислений.

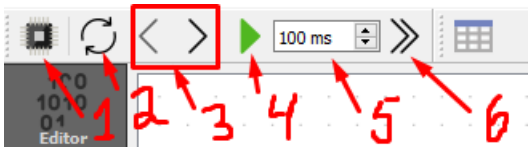
Начальные значения {x 1 , y 1 , z 1 } расположить в регистрах a2, a3, a4; значения {x 2 , y 2 , z 2 } расположить в регистрах a5, a6, a7; значения констант a, b, c расположить в регистрах s0, s1, s2. Результаты вычисления {r 1 , r 2 } записать в регистры a1, a2.

В исходном коде обязательно должны быть употреблены следующие псевдоинструкции: call (ровно 1 раз), ret (ровно 1 раз), mv (как минимум 1 раз), li (как минимум 2 раза: 1 раз – преобразующаяся в две инструкции; 1 раз – преобразующаяся в одну инструкцию).

Моделируемые вычисления (формула, входные данные, результаты) должны выводиться в консоль.

Основные теоретические положения.

Основные возможности эмулятора Ripes:



- Меню настройки процессора (*цифра 1*)
- Сброс состояния процессора, памяти, регистров и тд (*цифра 2*)
- Пошаговое выполнение (один такт вперёд или один такт назад) (*цифра 3*)
- Запуск в демонстрационном режиме, где каждый следующий такт выполняется через заданное количество времени (*цифра 4*)
- Установка временного интервала между тактами в демонстрационном режиме (*цифра 5*)
- “Моментальное” выполнение программы без задержек между тактами (*цифра 6*)

Системные вызовы:

1. PrintInt (a7=1): выводит число в регистре a0 в консоль в десятичной системе счисления
2. PrintString (a7=4): выводит строку, заканчивающуюся символом конца строки, адрес строки считывается из регистра a0
3. PrintChar (a7=11): выводит младший байт регистра a0 в консоль как символ
4. Write (a7=64): выводит в файловый дескриптор, номер которого записан в a0, строку с адреса, записанного в a1, длиною a2. В конце в a0 записывается количество записанных байт

5. Exit (a7=93): завершает программу с кодом возврата в регистре a0

Программа на Ассемблере состоит из директив (рассматриваются на этапе трансляции), инструкций (выполняются при запуске программы) и данных. Все они хранятся в соответствующих разделах в соответствии с их назначением. Разделы определяются с помощью директив. Основными разделами являются:

1. Раздел TEXT Раздел, доступный только для чтения, содержит фактические инструкции программы.

Синтаксис определения: .text

Описание. Этот раздел также известен как сегмент кода или просто текстовый сегмент программы. Он содержит исполняемые инструкции, которые не могут быть изменены во время выполнения. Любая попытка сохранить что-то в раздел text выдаст ошибку “Сегментация”, и программа немедленно завершится. Сегмент кода может в дополнение к инструкциям также содержать константы.

2. Раздел DATA Раздел, доступный для чтения и записи, содержит данные для переменных программы.

Синтаксис: .data Переменные

Описание. Раздел (сегмент) .data содержит инициализированные статические переменные, которые являются глобальными или статическими локальными переменными.

Инструкции и псевдо-инструкции:

Многие команды программ на ассемблере RISC-V не используют три аргумента, так как являются псевдоинструкциями. Это означает, что они являются сокращениями для других инструкций.

Пример.

Псевдоинструкция li rd, x5,0x123456 (загрузка большого значения в регистр) преобразуется ассемблером в 2 инструкции:

lui rd, 0x123 (загрузить константу в старшие биты 31-12 регистра)

addi rd, rd, 0x456 (сложение того же регистра с младшими 12 битами).

Выполнение работы.

1. Процедура calc_expression

Разработана процедура, вычисляющая выражение для целочисленных 32-битных входных переменных согласно варианту. Внутри вычисляется выражение для двух наборов данных {x1, y1, z1} и {x2, y2, z2}. Использованы инструкции xor (исключающее «или»), and (побитовое «и»), sll (побитовый сдвиг влево), а также псевдоинструкции neg (обратное число) и ret для возвращения из процедуры.

2. Основная программа

Состоит из двух разделов. В разделе data хранятся строки, которые впоследствии будут выводиться в консоль.

В разделе text содержится код программы. С помощью директивы equ определяются константы a, b, c, далее они загружаются в регистры инструкцией addi. Псевдо-инструкцией li в регистры загружаются значения для двух наборов {x1, y1, z1} и {x2, y2, z2}, для больших чисел данная псевдо-инструкция преобразуется в две: lui (загружает константу в старшие биты регистра) и addi (складывает тот же регистр с младшими 12-ю битами). Далее с помощью псевдо-инструкции la в регистры загружаются адреса строк для печати, они выводятся с помощью системного вызова printString. Символы и числа для печати загружаются в регистр a0 после чего выводятся системными вызовами PrintChar и PrintInt соответственно. Процедура calc_expression вызывается псевдо-инструкцией call, которая преобразуется в две инструкции: auipc и jalr. После вывода результатов, полученных в процедуре, программа завершается системным вызовом Exit.

Результаты отладки в пошаговом режиме под управлением отладчика с фиксацией содержимого используемых регистров и ячеек памяти до и после выполнения команд представлены в таблице 1. Тестирование программы на 3-х наборах для {x1, y1, z1} и 3-х наборах для {x2, y2, z2} представлены в таблице 2. Исходный код в приложении А.

Таблица 1 – результаты отладки программы в пошаговом режиме

Адрес инстр.	(Псевдо-) инстр.	Инструкция(и)	16-ричный код инстр.	Содержимое регистров и ячеек памяти	
				до вып. инстр.	после вып. инстр.
0		addi x8 x0 17	01100413	x8 s0 0x00000000	x8 s0 0x00000011
4		addi x9 x0 8	00800493	x9 s1 0x00000000	x9 s1 0x00000008
8		addi x18 x0 9	00900913	x18 s2 0x00000000	x18 s2 0x00000009
c	li a2, -17	addi x12 x0 -17	fef00613	x12 a2 0x00000000	x12 a2 0xffffffef
10	li a3, 5695	lui x13 0x1	000016b7	x13 a3 0x00000000	x13 a3 0x00001000
14		addi x13 x13 1599	63f68693	x13 a3 0x00001000	x13 a3 0x0000163f
18	li a4, 9	addi x14 x0 9	00900713	x14 a4 0x00000000	x14 a4 0x00000009
1c	li a5, -132	addi x15 x0 -132	f7c00793	x15 a5 0x00000000	x15 a5 0xffffff7c
20	li a6, -11	addi x16 x0 -11	ff500813	x16 a6 0x00000000	x16 a6 0xfffffff5
24	li a7, 310	addi x17 x0 310	13600893	x17 a7 0x00000000	x17 a7 0x00000136
28	mv s4, a7	addi x20 x17 0	00088a13	x20 s4 0x00000000 x17 a7 0x00000136	x20 s4 0x00000136 x17 a7 0x00000136
2c	la a0, formula	auipc x10 0x10000	10000517	x10 a0 0x00000000	x10 a0 0x1000002c
30		addi x10 x10 -44	fd450513	x10 a0 0x1000002c	x10 a0 0x10000000
34		addi x17 x0 4	00400893	x17 a7 0x00000136	x17 a7 0x00000004
38		ecall	00000073	x2 sp 7ffffff0	x2 sp 7ffffff0
3c	mv a0, a2	addi x10 x12 0	00060513	x10 a0 0x10000000 x12 a2 0xffffffef	x10 a0 0xffffffef x12 a2 0xffffffef
40		addi x17 x0 1	00100893	x17 a7 0x00000004	x17 a7 0x00000001
44		ecall	00000073	x2 sp 7ffffff0	x2 sp 7ffffff0
e4	call calc_expression	auipc x1 0x0 <_start>	00000097	x1 ra 0x00000000	x1 ra 0x000000e4
e8		Jalr x1 x1 72	048080e7	x1 ra 0x000000e4	x1 ra 0x000000ec

12c	neg s1, s1	sub x9 x0 x9	409004b3	x9 s1 0x00000008	x9 s1 0xfffffff8
130		xor x14 x14 x9	00974733	x14 a4 0x00000009 x9 s1 0xfffffff8	x14 a4 0xfffffff1 x9 s1 0xfffffff8
134		and x12 x12 x18	01267633	x12 a2 0xffffffef x18 s2 0x00000009	x12 a2 0x00000009 x18 s2 0x00000009
138		xor x14 x14 x12	00c74733	x14 a4 0xfffffff1 x12 a2 0x00000009	x14 a4 0xfffffff8 x12 a2 0x00000009
13c		sll x13 x13 x8	008696b3	x13 a3 0x0000163f x8 s0 0x00000011	x13 a3 0x2c7e0000 x8 s0 0x00000011
140		and x11 x14 x13	00d775b3	x11 a1 0x00000000 x14 a4 0xfffffff8 x13 a30x2c7e0000	x11 a1 0x2c7e0000 x14 a4 0xfffffff8 x13 a30x2c7e0000
158	ret	jalr x0 x1 0	00008067	x1 ra 0x000000ec	x1 ra 0x000000ec
120		addi x10 x0 1	00100513	x10 a0 0x00000000a	x10 a0 0x000000001
124		addi x17 x0 93	05d00893	x17 a7 0x0000000b	x17 a7 0x0000005d
128		ecall	00000073	x2 sp 7ffffff0	x2 sp 7ffffff0

Таблица 2 – результаты тестирования

Входные данные	Результаты
$x_1 = 10$ $y_1 = 5$ $z_1 = 6$ $x_2 = 3703214$ $y_2 = 65$ $z_2 = 1193046$	Formula: ((z ^ (-8)) ^ (x & 9)) & (y << 17) Input data: {x1, y1, z1} = 10 5 6 {x2, y2, z2} = 3703214 65 1193046 Results: 655360 8388608
$x_1 = 4761$ $y_1 = 7968$ $z_1 = 8888$ $x_2 = -88$ $y_2 = -17$ $z_2 = -6$	Formula: ((z ^ (-8)) ^ (x & 9)) & (y << 17) Input data: {x1, y1, z1} = 4761 7968 8888 {x2, y2, z2} = -88 -17 -6 Results: 1044381696 0
$x_1 = 18$ $y_1 = 11$ $z_1 = 8$ $x_2 = 31$ $y_2 = 54$ $z_2 = -12$	Formula: ((z ^ (-8)) ^ (x & 9)) & (y << 17) Input data: {x1, y1, z1} = 18 11 8 {x2, y2, z2} = 31 54 -12 Results: 1441792 0
$x_1 = -17$ $y_1 = 5695$	Formula: ((z ^ (-8)) ^ (x & 9)) & (y << 17) Input data:

$z_1 = 9$ $x_2 = -132$ $y_2 = -11$ $z_2 = 310$	{x1, y1, z1} = -17 5695 9 {x2, y2, z2} = -132 -11 310 Results: 746455040 -1441792

Выводы.

В ходе выполнения лабораторной работы установлена, настроена и изучена среда эмулятора Ripes. Получены знания об архитектуре RISC-V, базовом наборе инструкций, псевдо-инструкциях и регистрах. Разработана простая программа на ассемблере.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ.

Название файла lb5.s:

```
.text                       # раздел с инструкциями программы
.global _start
_start:
    .equ a, 17              # задаются константы
    .equ b, 8
    .equ c, 9
    addi s0, x0, a          # константы a, b, c размещаются
    addi s1, x0, b          # в регистры s0, s1
    addi s2, x0, c          # и s2 соответственно
    li a2, -17              # размещение переменных
    li a3, 5695             # x1, y1, z1 в регистры
    li a4, 9                # a2, a3, a4 соответственно
    li a5, -132             # размещение переменных
    li a6, -11              # x2, y2, z2 в регистры
    li a7, 310              # a5, a6, a7 соответственно
    mv s4, a7               # сохранение значения из a7
    la a0, formula          # вывод строки по адресу в a0
    addi a7, x0, 4          # с помощью системного вызова PrintString
    ecall                   # (a7=4)
    mv a0, a2               # перенос числа из a2 в a0
    addi a7, x0, 1          # вывод его в 10-чной сс системным вызовом
    ecall                   # PrintInt (a7=1)
    li a0, 32               # в a0 ascii код пробела
    addi a7, x0, 11         # вывод символа системным вызовом
    ecall                   # PrintChar (a7=11)
    mv a0, a3               # вывод числа из a3
    addi a7, x0, 1
    ecall
    li a0, 32
    addi a7, x0, 11
    ecall
    mv a0, a4               # вывод числа из a4
    addi a7, x0, 1
    ecall
    li a0, 10               # вывод символа переноса строки
    addi a7, x0, 11
    ecall
    la a0, data             # вывод строки
    addi a7, x0, 4
    ecall
    mv a0, a5               # вывод числа из a5
    addi a7, x0, 1
    ecall
    li a0, 32
    addi a7, x0, 11
    ecall
    mv a0, a6               # вывод числа из a6
    addi a7, x0, 1
    ecall
    li a0, 32
    addi a7, x0, 11
    ecall
```

```
    mv a7, s4             # восстановление значения в a7
    mv a0, a7             # вывод числа из a7
    addi a7, x0, 1
    ecall
    li a0, 10
    addi a7, x0, 11
    ecall
    mv a7, s4
    call calc_expression          # вызов процедуры calc_expression
    la a0, results                # вывод строки
    addi a7, x0, 4
    ecall
    mv a0, a1                     # вывод числа из a1
    addi a7, x0, 1
    ecall
    li a0, 10
    addi a7, x0, 11
    ecall
    mv a0, a2                     # вывод числа из a2
    addi a7, x0, 1
    ecall
    addi a0, x0, 1                # завершение программы системным
вызовом
    addi a7, x0, 93               # Exit (a7=93) с кодом возврата в
регистре a0
    ecall

calc_expression:
    neg s1, s1            # теперь в s1 находится (-b)
# вычисление выражения для {x1, y1, z1}
    xor a4, a4, s1        # теперь в a4 находится (z1 ^ (-b))
    and a2, a2, s2        # теперь в a2 находится (x1 & c)
    xor a4, a4, a2        # теперь в a4 находится ((z1 ^ (-b)) ^ (x1 &
c))
    sll a3, a3, s0        # теперь в a3 находится (y1 << a)
    and a1, a4, a3        # теперь в a1 находится ((z1 ^ (-b)) ^ (x1 &
c)) & (y1 << a)
# вычисление выражения для {x2, y2, z2}
    xor a7, a7, s1        # теперь в a7 находится (z2 ^ (-b))
    and a5, a5, s2        # теперь в a5 находится (x2 & c)
    xor a7, a7, a5        # теперь в a7 находится ((z2 ^ (-b)) ^ (x2 &
c))
    sll a6, a6, s0        # теперь в a6 находится (y2 << a)
    and a2, a7, a6        # теперь в a2 находится ((z2 ^ (-b)) ^ (x2 &
c)) & (y2 << a)
    ret                   # возврат из процедуры

.data                     # раздел с данными для переменных программы
formula: .asciz "Formula: ((z ^ (-8)) ^ (x & 9)) & (y << 17)\nInput
data:\n{x1, y1, z1} = "
data: .asciz "{x2, y2, z2} = "
results: .asciz "Results:\n"
```