**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

**отчет**

**по лабораторной работе №7**

**по дисциплине «Организация ЭВМ и систем»**

**Тема: Взаимодействие с внешними компонентами.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Сабалиров М.З. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Молодцев Д.А |

Санкт-Петербург

2024

**Цель работы**

1. Освоить программирование на ассемблере RISC-V взаимодействия с внешними устройствами (ввода и вывода).

**Задание:**

1. Для выполнения работы требуется реализовать ряд комбинационных функций на ассемблере. Каждое вычисленное значение функции определяет состояние светодиода: 0 – не горит, 1 – горит. Значение переменных определяется переключателями: 0 – выкл., 1 – вкл.

Вариант 18:

|  |  |
| --- | --- |
| 18 | 2,5,9 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X1 | X2 | X3 | X4 | Y2 | Y5 | Y9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |

**Основные теоретические положения:**

Одной из важнейших функций, которые решаются микроконтроллером, является взаимодействие с внешними элементами. Для этого в микросхемах реализуются механизмы аппаратной передачи данных. Самым простым механизмом является интерфейс GPIO (General-Purpose Input/Output), в котором основные операции - установка на контакте микросхемы определенного логического уровня и считывание сигнала с контакта.

Эмулятор Ripes имеет возможность эмуляции внешних устройств по интерфейсу GPIO. Обеспечивается работа со следующими электронными компонентами:

Светодиодные матрицы (LED Matrix)

Клавиатура со стрелками (D-Pad)

Переключатели (Switches)

Клавиатура и переключатели используются как устройства ввода, а светодиоды – как устройство вывода.

Для создания компонентов требуется перейти на вкладку I/O и добавить желаемые устройства.

По умолчанию создаются следующие компоненты:

D-Pad (не изменяется)

Switches на 8 переключателей

LED Matrix 35×25 светодиодов, размер которых равен 8

На рисунке представлен пример созданных и модифицированных компонентов: матрица светодиодов 8×8, размер которых равен 32, клавиатура и 4 переключателя.

Светодиоды для матрицы управляются независимо и могут показывать любой цвет. Пример горящих светодиодов для матрицы 32×32 показан на рисунке.

Для созданных компонентов определяются адреса в памяти процессора, по которым осуществляется взаимодействие с компонентами. Адреса указаны в виде кода для языка С, который можно скопировать для программ. Пример:

#ifndef RIPES\_IO\_HEADER

#define RIPES\_IO\_HEADER

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \* LED\_MATRIX\_0

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define LED\_MATRIX\_0\_BASE (0xf0000000)

#define LED\_MATRIX\_0\_SIZE (0x100)

#define LED\_MATRIX\_0\_WIDTH (0x8)

#define LED\_MATRIX\_0\_HEIGHT (0x8)

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \* D\_PAD\_0

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define D\_PAD\_0\_BASE (0xf0000114)

#define D\_PAD\_0\_SIZE (0x10)

#define D\_PAD\_0\_UP\_OFFSET (0x0)

#define D\_PAD\_0\_UP (0xf0000114)

#define D\_PAD\_0\_DOWN\_OFFSET (0x4)

#define D\_PAD\_0\_DOWN (0xf0000118)

#define D\_PAD\_0\_LEFT\_OFFSET (0x8)

#define D\_PAD\_0\_LEFT (0xf000011c)

#define D\_PAD\_0\_RIGHT\_OFFSET (0xc)

#define D\_PAD\_0\_RIGHT (0xf0000120)

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \* SWITCHES\_0

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define SWITCHES\_0\_BASE (0xf0000110)

#define SWITCHES\_0\_SIZE (0x4)

#define SWITCHES\_0\_N (0x4)

#endif // RIPES\_IO\_HEADER

Для всех компонентов определен Base адрес. Это – начальный адрес компонента. Для доступа к каждому последующему элементу требуется смещение на значение n, которое каждое для своего компонента. Так, для доступа к n-ному ключу требуется выполнить смещение n=x где x – порядковый номер ключа при отсчете с нуля. Для светодиодной матрицы n = y\*W+x , где W – ширина матрицы, x,y – номера строки и столбца, на пересечении которых находится светодиод.

Так, для того, чтобы зажечь светодиод нужно определить его адрес и записать по нему значение цвета в формате RGB. Для языка С это будет выглядеть следующим образом:

\*(led\_base + idx) = r << 16 | g << 8 | b;

где led\_base – начальный адрес матрицы,

idx – смещение,

r, g, b – байт, определяющий степень выраженности красного, зеленого и синего цветов соответственно.

Эта команда записывает по адресу (led\_base + idx) значение, определяемое как (r << 16 | g << 8 | b).

Пример кода для анимации:

.text

li a0 LED\_MATRIX\_0\_BASE # a0 = BASE

li a1 LED\_MATRIX\_0\_WIDTH # a1 = WIDTH

li a2 LED\_MATRIX\_0\_HEIGHT # a2 = HEIGHT

# t4 - индекс ряда (y)

# a4 - индекс пикселя в ряду (x)

# a3 - количество оставшихся в ряду пикселей (HEIGHT - y)

# t1 - количество пройденных пикселей в ряду (WIDTH - x)

# t5 = G составляющая цвета

# s1 = B составляющая цвета

# s0 - цвет светодиода (RGB)

# a5 - адрес светодиода

# t0 = 0xFF0000, a7 = 0xFF00 - маски для R и G цветов

# a6 = WIDTH << 2, t3 = WIDTH + HEIGHT - константы

# t6 - номер итерации для анимации

loop:

addi sp, sp, -16 # (sp -= 16)

# sp = 0x7FFFFFE0

sw s0, 12(sp) # \*(sp + 12) = s0

sw s1, 8(sp) # \*(sp + 8) = s1

mv t6, zero # t6 = 0

add t3, a2, a1 # t3 = a2 + a1 = WIDTH + HEIGHT

slli a6, a1, 2 # a6 = a1 << 2 = WIDTH << 2

lui a3, 16 # a3 = 16 << 12 = 0x10000

addi a7, a3, -256 # a7 = a3 - 256 = 0xFF00

lui t0, 4080 # t0 = 0xFF0000

init: # инициализация

mv t4, zero

mv t1, zero

mv t2, a0 # t1 = t4 = 0; t2 = BASE

nextRow: # обработать ряд

mv a4, zero # a4 = 0

slli a3, t1, 8

sub a3, a3, t1

divu a3, a3, a2

add a3, a3, t6

slli a3, a3, 8 # a3 =

# (((t1 \* 255)/HEIGHT)+t6)<<8

and t5, a3, a7 # t5 = a3 & a7 (G - ЦВЕТ)

mv a5, t2 # a5 = t2

mv a3, a1 # a3 = WIDTH

nextPixel: # задать цвет пикселя

divu s0, a4, a1

add s0, s0, t6 # s0 = (a4/a1) + t6

add s1, t4, a4

divu s1, s1, t3

add s1, s1, t6 # s1 = ((t4+a4)/t3)+t6

slli s0, s0, 16

and s0, s0, t0 # s0 &= 0xFF0000 (R - ЦВЕТ)

or s0, t5, s0 # s0 = (s0 << 16) & t0 | t5 (RG)

andi s1, s1, 255 # s1 &= 0xFF (B - ЦВЕТ)

or s0, s0, s1 # s0 |= s1 (RGB - ЦВЕТ)

sw s0, 0(a5) # \*(0+a5) = s0; ОКРАШИВАНИЕ

addi a3, a3, -1 # a3—-

addi a5, a5, 4 # a5 += 4

addi a4, a4, 255 # a4 += 0xFF

# если ряд не окончился окрашиваем следующий пиксель,

# иначе переходим на следующий ряд

bnez a3, nextPixel # if (a3 != 0) goto nextPixel

addi t6, t6, 1 # t6++

addi t1, t1, 1 # t1++

add t2, t2, a6 # t2 += a6

addi t4, t4, 255 # t4 += 0xFF

# если ряд не последний переходим на следующий ряд,

# иначе повтор с начала

bne t1, a2, nextRow # if (t1 != HEIGHT) goto nextRow

j init # restart

Данный код эквивалентен следующей программе на языке С:

#define W LED\_MATRIX\_0\_WIDTH

#define H LED\_MATRIX\_0\_HEIGHT

unsigned\* led\_base = LED\_MATRIX\_0\_BASE;

void main() {

unsigned v = 0;

while (1) {

for (int y = 0; y < H; y++) {

for (int x = 0; x < W; x++) {

char r = v + (0xFF \* x) / W;

char g = v + (0xFF \* y) / H;

char b = v + (0xFF \* (x + y)) / (W + H);

unsigned idx = y \* W + x;

\*(led\_base + idx) = r << 16 | g << 8 | b;

}

v++;

}

}

}

Таким образом переход по метке init обеспечивает работу бесконечного цикла, метка nextRow обозначает внешний цикл, а nextPixel - внутренний. Для вычисления цвета (регистр s0) и индекса светодиода (регистр a5) используется большое количество внутренних регистров из-за большого количества операций. Регистр a6 соответствует переменной v, а регистр a5 - переменной idx.

Для записи в память для установки цвета светодиода используется команда

SW rs2, imm12(rs1)

Эта команда записывает слово (32 бита) по адресу rs1+imm12.

Получение информации об устройстве осуществляется чтением нужного участка памяти. Так, для получения цвета светодиода под номером idx требуется прочитать область памяти \*(led\_base + idx). Тогда, 24-битное значение цвета светодиода на языке С можно получить с помощью команды

unsigned int color = \*(led\_base + idx)

В битах [16..23] расположен байт, определяющий выраженность красного цвета, в битах [8..15] - зеленого цвета, а в битах [0..7] (младший байт) - синего цвета.

Для чтения памяти используется команда

LW rd, imm12(rs1)

Эта команда осуществляет чтение 32 бит памяти по адресу rs1+imm12 и записывает результат в регистр rd. Существует команда LWU с аналогичным назначением и применением для получения данных в беззнаковом виде.

В качестве примера рассмотрим программу, выполняющую зажигание светодиодов в зависимости от того, какие переключатели включены. Конфигурация представлена на рисунке.

Рассмотрим задачу включения светодиодов, которые соответствуют ключам. Цвет первого в матрице светодиода - красный, второго - зеленый, третьего - синиц, а четвертого - смесь цветов остальных светодиодов. Если ключ светодиода выключен, то светодиод не горит (черный цвет). Для решения этой задачи требуется прочитать конфигурацию переключателей из памяти, определить цвета светодиодов и записать их в память, соответствующую светодиодам. Простейшая программа выглядит следующим образом:

li a0 LED\_MATRIX\_0\_BASE

li a1 LED\_MATRIX\_0\_WIDTH

li a2 LED\_MATRIX\_0\_HEIGHT

li a3 SWITCHES\_0\_BASE

li a4 SWITCHES\_0\_SIZE

# s0 - color[0]

# s1 - color[1]

# s2 - color[2]

# s3 - color[3]

# t0 - switch[0]

# t1 - switch[1]

# t2 - switch[2]

# t3 - switch[3]

# a5 - слово состояния ключей

loop: # бесконечный цикл

lw a5, 0[a3] # чтение состояния переключателя

mv s0,zero

mv s1,zero

mv s2,zero

mv s3,zero # обнуление цветов

andi t0,a5,1

andi t1,a5,2

andi t2,a5,4

andi t3,a5,8 # перенос битов переключателя в

# отдельные регистры

beqz t0, skipColor0

lui s0,4080 # цвет светодиода 0

skipColor0:

beqz t1, skipColor1

li s1, 65280 # цвет светодиода 1

skipColor1:

beqz t2, skipColor2

li s2,255 # цвет светодиода 2

skipColor2:

beqz t3, skipColor3 # если в t-регистре 0, не окрашиваем

or s3,s3,s0

or s3,s3,s1

or s3,s3,s2 # цвет светодиода 3

skipColor3:

sw s0, 0(a0) # пишем цвета в память для окраса

sw s1, 4(a0)

sw s2, 8(a0)

sw s3, 12(a0)

j loop # бесконечный цикл

**Выполнение работы**

0. Среда разработки была настроена

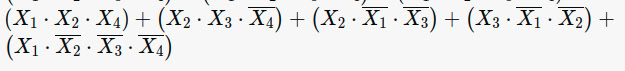
1. Исходные функции были переведены из табличного вида в алгебраический:

Y2: ((~ x\_1) ∧ (~ x\_2) ∧ x\_3 ∧ (~ x\_4)) ∨ ((~ x\_1) ∧ (~ x\_2) ∧ x\_3 ∧ x\_4) ∨ ((~ x\_1) ∧ x\_2 ∧ (~ x\_3) ∧ (~ x\_4)) ∨ ((~ x\_1) ∧ x\_2 ∧ (~ x\_3) ∧ x\_4) ∨ ((~ x\_1) ∧ x\_2 ∧ x\_3 ∧ (~ x\_4)) ∨ (x\_1 ∧ (~ x\_2) ∧ (~ x\_3) ∧ (~ x\_4)) ∨ (x\_1 ∧ x\_2 ∧ (~ x\_3) ∧ x\_4) ∨ (x\_1 ∧ x\_2 ∧ x\_3 ∧ (~ x\_4)) ∨ (x\_1 ∧ x\_2 ∧ x\_3 ∧ x\_4)

Y5: ((~ x\_1) ∧ (~ x\_2) ∧ (~ x\_3) ∧ x\_4) ∨ ((~ x\_1) ∧ (~ x\_2) ∧ x\_3 ∧ (~ x\_4)) ∨ ((~ x\_1) ∧ (~ x\_2) ∧ x\_3 ∧ x\_4) ∨ ((~ x\_1) ∧ x\_2 ∧ (~ x\_3) ∧ x\_4) ∨ ((~ x\_1) ∧ x\_2 ∧ x\_3 ∧ x\_4) ∨ (x\_1 ∧ (~ x\_2) ∧ (~ x\_3) ∧ (~ x\_4)) ∨ (x\_1 ∧ (~ x\_2) ∧ x\_3 ∧ x\_4) ∨ (x\_1 ∧ x\_2 ∧ (~ x\_3) ∧ (~ x\_4)) ∨ (x\_1 ∧ x\_2 ∧ (~ x\_3) ∧ x\_4) ∨ (x\_1 ∧ x\_2 ∧ x\_3 ∧ (~ x\_4)) ∨ (x\_1 ∧ x\_2 ∧ x\_3 ∧ x\_4)

Y9: ((~ x\_1) ∧ x\_2 ∧ (~ x\_3) ∧ (~ x\_4)) ∨ ((~ x\_1) ∧ x\_2 ∧ (~ x\_3) ∧ x\_4) ∨ ((~ x\_1) ∧ x\_2 ∧ x\_3 ∧ x\_4) ∨ (x\_1 ∧ (~ x\_2) ∧ (~ x\_3) ∧ x\_4) ∨ (x\_1 ∧ (~ x\_2) ∧ x\_3 ∧ x\_4)

2. Функции были упрощенны:

Y2:

  
 Y5:

Y9:

3. Код из теоритических положений был доработан:

snez t0, t0

snez t1, t1

snez t2, t2

snez t3, t3

Добавлено после получения значений переключателей, таким образом в t0-t3 всегда 0 или 1 (snez set if ≠ zero)

4. Формулы были реализованы в программный код по средствам вычислений в регистрай t4-t6.

В t4 хранится результат всей формулы, в t5 текущее слогаемое (в контексте булевой алгебры, под сложением подразумевается логическое или), в t6 инвертированные, с помощью seqz значения (set if = zero).

5. Вычисления были реализованы в процедурах Y2, Y5, Y9

6. Тестирование:

Таблица 1. Тестирование

|  |  |  |  |
| --- | --- | --- | --- |
| № | Исходные данные | Ожидаемые результаты | Полученный результат |
| 1 | X1 = X2 = X3 = X4 = 0 | Все светодиоды выключенны |  |
| 2 | X1 = X2 = X4 = 0, X3 = 1 | Горят красный, зеленый |  |
| 3 | X2 = X4 = 1, X1 = X3 = 0 | Горят все |  |
| 4 | X1 = X4 = 1, X2 = X3 = 0 | Горит синий, 4-й горит синим |  |

4. Исходный код программы см. в **Приложении.**

**Вывод**

В результате выполнения лабораторной работы были получены навыки работы с ассемблером Risc-v на эмуляторе Ripes. Разработана программа, которая включает/выключает светодиоды в соответствии с таблицей истинности.

**ПРИЛОЖЕНИЕ**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

li a0 LED\_MATRIX\_0\_BASE

li a1 LED\_MATRIX\_0\_WIDTH

li a2 LED\_MATRIX\_0\_HEIGHT

li a3 SWITCHES\_0\_BASE

li a4 SWITCHES\_0\_SIZE

# s0 - color[0]

# s1 - color[1]

# s2 - color[2]

# s3 - color[3]

# t0 - switch[0]

# t1 - switch[1]

# t2 - switch[2]

# t3 - switch[3]

# a5 - слово состояния ключей

loop: # бесконечный цикл

lw a5, 0[a3] # чтение состояния переключателя

mv s0,zero

mv s1,zero

mv s2,zero

mv s3,zero # обнуление цветов

andi t0,a5,1

andi t1,a5,2

andi t2,a5,4

andi t3,a5,8 # перенос битов переключателя в

# отдельные регистры

snez t0, t0

snez t1, t1

snez t2, t2

snez t3, t3

jal ra, Y2

beqz t4, skipColor0

lui s0,4080 # цвет светодиода 0

skipColor0:

jal ra, Y5

beqz t4, skipColor1

li s1, 65280 # цвет светодиода 1

skipColor1:

jal ra, Y9

beqz t4, skipColor2

li s2,255 # цвет светодиода 2

skipColor2:

beqz t3, skipColor3 # если в t-регистре 0, не окрашиваем

or s3,s3,s0

or s3,s3,s1

or s3,s3,s2 # цвет светодиода 3

skipColor3:

sw s0, 0(a0) # пишем цвета в память для окраса

sw s1, 4(a0)

sw s2, 8(a0)

sw s3, 12(a0)

j loop # бесконечный цикл

Y2:

and t4, t0, t1

and t4, t4, t3

and t5, t1, t2

seqz t6, t3

and t5, t5, t6

or t4, t4, t5

mv t5, t1

seqz t6, t0

and t5, t5, t6

seqz t6, t2

and t5, t5, t6

or t4, t4, t5

mv t5, t2

seqz t6, t0

and t5, t5, t6

seqz t6, t1

and t5, t5, t6

or t4, t4, t5

mv t5, t0

seqz t6, t1

and t5, t5, t6

seqz t6, t2

and t5, t5, t6

seqz t6, t3

and t5, t5, t6

or t4, t4, t5

jr ra

Y5:

and t4, t0, t1

seqz t6, t0

and t5, t6, t3

or t4, t4, t5

and t5, t3, t2

or t4, t4, t5

seqz t6, t2

and t5, t0, t6

seqz t6, t3

and t5, t5, t6

or t4, t4, t5

seqz t6, t0

and t5, t2, t6

seqz t6, t1

and t5, t5, t6

or t4, t4, t5

jr ra

Y9:

and t4, t0, t3

seqz t6, t1

and t4, t4, t6

and t5, t1, t3

seqz t6, t0

and t5, t5, t6

or t4, t4, t5

seqz t6, t0

and t5, t1, t6

seqz t6, t2

and t5, t5, t6

or t4, t4, t5

jr ra