МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФГБОУ ВО «АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ЦИФРОВЫХ ТЕХНОЛОГИЙ, ЭЛЕКТРОНИКИ И ФИЗИКИ (ИЦТЭФ)

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И ЭЛЕКТРОНИКИ (ВТиЭ)

**Отчет по лабораторной работе № 1**

по курсу “Микропроцессорные системы”

**“Изучение микроконтроллеров семейства Intel 8051”**

Выполнил студент 506 гр.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.В. Осипенко

Проверил: ст. преп. кафедры ВТиЭ.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Белозерских

Лабораторная работа защищена

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Цель работы

Получение навыков программирования микропроцессорных систем на базе микроконтроллеров MCS-51, при помощи среды разработки ProView32 и программатора ProAtMic.

# Задача

Изучить основы программирования микропроцессорных систем на базе микроконтроллера Atmel AT89C4051. Написать и отладить программы с помощью среды программирования и произвести программирование контроллера. Работоспособность программ проверить на лабораторной установке.

# Задание 1

Написать программу сложения числа, находящегося во второй половине внутренней памяти, с числом 5. Результат поместить во внешнюю память.

## Алгорит:

НАЧАЛО.

1. Записываем значение из внут. памяти в рег. А.

2. Прибавляем 5Н к регистру А.

3. Заносим внеш. адрес в рег. DPTR.

4. Записываем значение рег. А во внеш. память по адресу в рег. DPTR.

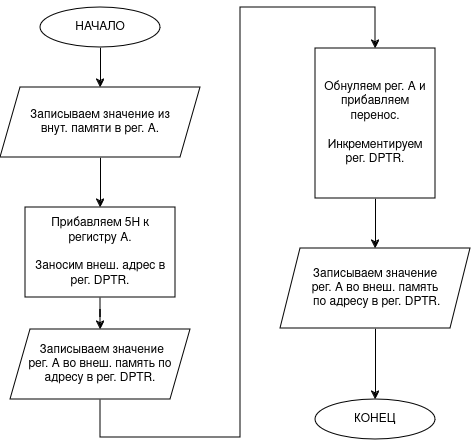
5. Обнуляем рег. А и прибавляем перенос.

6. Инкрементируем рег. DPTR.

7. Записываем значение рег. А во внеш. память по адресу в рег. DPTR.

КОНЕЦ.

## Блок схема:



## Код:

MAIN SEGMENT CODE

CSEG AT 0

LJMP start

USING 0

RSEG MAIN

start:

MOV A, 40H ; записываем значение по адресу 0х40 внутренней памяти в регистр А

ADD A, #05H ; прибавляем в регистр А значение 0х5

MOV DPTR, #0020H ; заносим значение адреса внешней памяти в регистр DPTR

MOVX @DPTR, A ; записываем регистр А во внешнюю память по адресу в регистре DPTR

CLR A ; зануляем А

ADDC A, 0H ; записываем результат переноса в регистр А

INC DPTR ; увеличиваем значение рег. DPTR

MOVX @DPTR, A ; записываем рег. А во внешнюю память по адресу в регистре DPTR

JMP start

END

## Вывод:

Задание выполнено успешно. В рамках работы были получены навыки работы с внутренней и внешней памятью контроллера. Произведено считывание числа из внутренней памяти, прибавление к нему числа 5Н и запись результата во внешнюю память.

# Задание 2

Написать программу вычитания из числа, находящегося во внешней памяти данных МК, числа 5. Результат поместить во внутреннюю память данных, используя косвенную адресацию.

## Алгорит:

НАЧАЛО.

1. Читаем число из внешней памяти в регистр A.

2. Вычитаем 5Н из рег. А.

3. Заносим адрес внут. памяти в рег. R0.

4. Записываем значение рег. А во внут. память с помощью рег. R0.

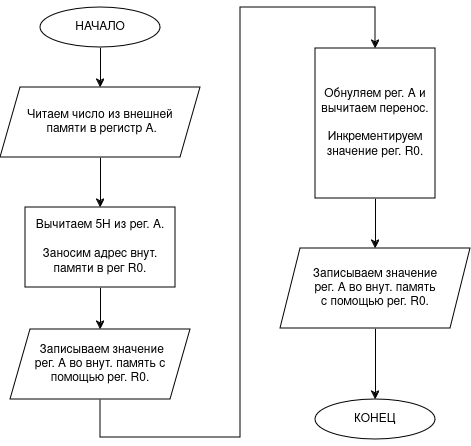
5. Обнуляем рег. А и вычитаем перенос.

6. Инкрементируем значение рег. R0.

7. Записываем значение рег. А во внут. память с помощью рег. R0.

КОНЕЦ.

## Блок схема:



## Код:

MAIN SEGMENT CODE

CSEG AT 0L

JMP START

USING 0

RSEG MAIN

START:

; Загрузка значения из памяти по адресу 0020H в регистр A

MOV DPTR, #020H

MOVX A, @DPTR

; Вычитание 5 из значения в регистре A

CLR C

SUBB A, #5H

; Сохранение результата в память по адресу 40H

MOV R0, #40H

MOV @R0, A

; Установка значения A в 00H

MOV A, #0H

; Вычитаем перенос

SUBB A, #0H

; Увеличение значения в регистре R0 на 1

INC R0

; Сохранение значения A в память по новому адресу, указанному в R0

MOV @R0, A

; Переход к метке START

JMP START

END

## Вывод:

Задание выполнено успешно. В рамках работы были получены навыки работы с внутренней и внешней памятью контроллера. Произведено считывание числа из внешней памяти, вычитания из него числа 5Н и запись результата во внутреннюю память.

# Задание 3

Написать программу умножения двух трехбайтных чисел. Первое число находится во внешней памяти, второе число принимается побайтно из порта 2 микроконтроллера. Результата умножения вывести в последовательный порт (UART). Скорость 38400 Гц и режим работы UART: 9 бит. Обязательное использование подпрограмм и символических переменных в отдельном файле проекта.

## Алгорит:

Функция main:

НАЧАЛО.

1. Вызов READ\_PORT.

2. Вызов EVAL\_NUMS.

3. Вызов SUM\_NUMS.

4. Вызов OUT\_UART.

КОНЕЦ.

Функция SUM\_NUMS:

НАЧАЛО.

1. В 0-й байт результата записываем 0-й байт первого слагаемого.

2. В 1-й байт результата записываем сумму 1-го и 0-го байтов первого и второго слагаемых.

3. В 2-й байт результата записываем сумму 2-го, 1-го и 0-го байтов первого, второго и третьего слагаемых и прибавляем значение переноса.

4. В 3-й байт результата записываем сумму 3-го, 2-го и 1-го байтов первого, второго и третьего слагаемых и прибавляем значение переноса.

5. В 4-й байт результата записываем сумму 3-го и 2-го байтов второго и третьего слагаемых и прибавляем значение переноса.

6. В 5-й байт результата записываем сумму 3-го байта третьего слагаемого и прибавляем значение переноса.

КОНЕЦ.

Функция OUT\_UART:

НАЧАЛО.

1. Настраиваем UART, выставляя заданные значения.

2. Устанавливаем i = 0

3. Вывод i-й байт результата в буфер

4. Инкрементируем i

5. Если i < 6, то переходим в пункт 3.

КОНЕЦ.

Функция READ\_PORT:

НАЧАЛО.

1. Устанавливаем i = 0.

2. Читаем i-й бит 2-го числа

3. Инкрементируем i

4. Если i < 3, то переходим в пункт 2.

КОНЕЦ.

Функция EVAL\_NUMS:

НАЧАЛО.

1. Устанавливаем i = 0.

2. Устанавливаем j = 0.

3. Вычисляем произведение i-го и j-го байтов 1-го и 2-го числа.

4. Записываем результат в стэк.

5. Инкрементируем j.

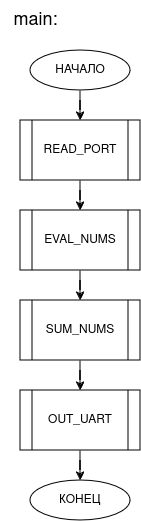
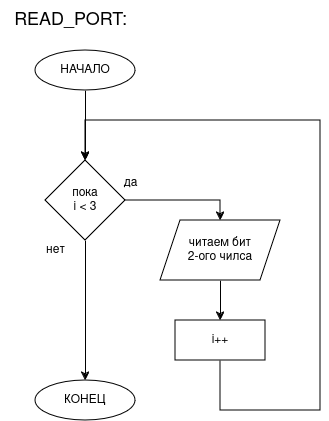
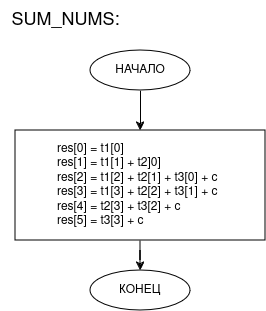
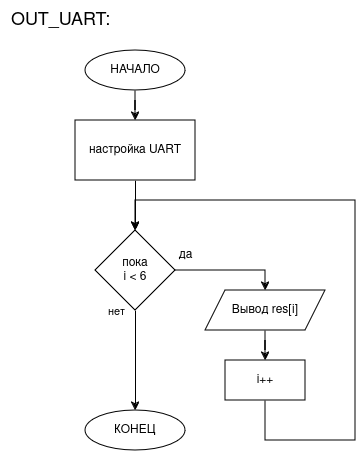
6. Если j < 3, то переходим в пункт 3.

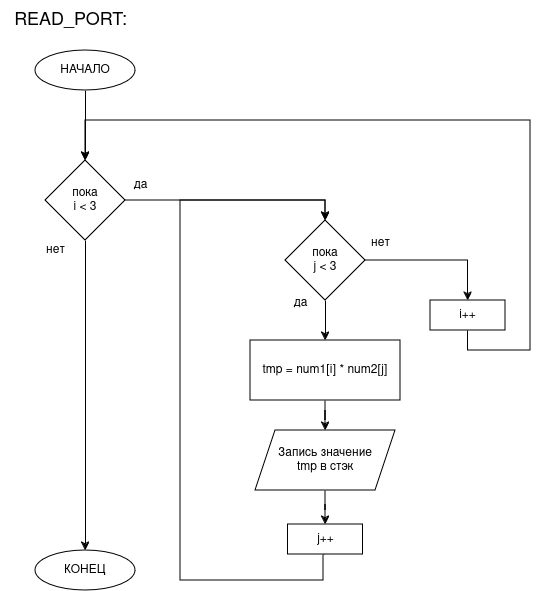
7. Инкрементируем i.

8. Если i < 3, то переходим в пункт 2.

КОНЕЦ.

## Блок схема:



## Код:

<Файл main.asm>

EXTRN DATA (TMP1, TMP2, TMP3)

EXTRN XDATA (NUM1, NUM2, RES)

EXTRN CODE (READ\_PORT, EVAL\_NUMS, SUM\_NUMS, OUT\_UART)

MAIN SEGMENT CODE

CSEG AT 0

LJMP start

USING 0

START:

; Вызов функции чтения данных из порта

CALL READ\_PORT

; Вызов функции обработки числовых данных

CALL EVAL\_NUMS

; Вызов функции сумирования чисел

CALL SUM\_NUMS

; Вызов функции вывода результата через UART

CALL OUT\_UART

; Бесконечная петля предотвращения выхода из программы

JMP $

END

<Файл vars.asm>

PUBLIC TMP1, TMP2, TMP3, NUM1, NUM2, RES

XVARS SEGMENT XDATA

VARS SEGMENT DATA

RSEG VARS

TMP1: DS 4

TMP2: DS 4

TMP3: DS 4

RSEG XVARS

NUM1: DS 3

NUM2: DS 3

RES: DS 6

END

<Файл fns.asm>

PUBLIC READ\_PORT, EVAL\_NUMS, SUM\_NUMS, OUT\_UART

EXTRN DATA (TMP1, TMP2, TMP3)

EXTRN XDATA (NUM1, NUM2, RES)

FNS SEGMENT CODE

RSEG FNS

; Читаем число из порта

READ\_PORT:

; Устанавливаем счетчик

MOV R3, #03H

; Устанавливаем начальный адрес числа в внешней памяти

MOV DPTR, #NUM2

READ\_LOOP:

; Читаем побайтно число из 2-го порта

MOV A, P2

MOVX @DPTR, A

INC DPTR

DJNZ R3, READ\_LOOP

RET

; Функция вычисляет 3-и слагаемых для последующего произведения

EVAL\_NUMS:

; Подготавливаем стек и счетчики

POP ACC

MOV R4, A

POP ACC

MOV R5, A

MOV P2, #00H

MOV R0, #NUM1-1

MOV R6, #03H

O\_LOOP:

; Умножение i-го байта первого числа на 0-й байт второго числа

INC R0

MOV R1, #NUM2

MOVX A, @R1

MOV B, A

MOVX A, @R0

MUL AB

PUSH ACC

MOV A, B

PUSH ACC

MOV R7, #02H

S\_LOOP:

; Умножение i-го байта первого числа на оставшиеся байты второго

INC R1

MOVX A, @R1

MOV B, A

MOVX A, @R0

MUL AB

MOV R3, B

MOV B, A

POP ACC

ADD A, B

PUSH ACC

MOV A, R3

ADDC A, #00H

PUSH ACC

DJNZ R7, S\_LOOP

DJNZ R6, O\_LOOP

MOV A, R5

PUSH ACC

MOV A, R4

PUSH ACC

RET

; Функция осуществляет получение итогового числа путем сложения трех слагаемых

SUM\_NUMS:

MOV P2, #00H

MOV R0, #RES

; Получаем 0-й байт числа

MOV A, TMP1

MOVX @R0, A

; Вычисляем 1-й байт числа

INC R0

MOV B, TMP1 + 1

MOV A, TMP2

ADD A, B

MOVX @R0, A

; Вычисляем 2-й байт числа

INC R0

MOV B, TMP1 + 2

MOV A, TMP2 + 1

ADDC A, B

MOV B, A

MOV A, TMP1 + 3

ADDC A, #0

MOV TMP1 + 3, A

MOV A, TMP3

ADDC A, B

MOVX @R0, A

; Вычисляем 3-й байт числа

INC R0

MOV B, TMP1 + 3

MOV A, TMP2 + 2

ADDC A, B

MOV B, A

MOV A, TMP2 + 3

ADDC A, #0

MOV TMP2 + 3, A

MOV A, TMP3 + 1

ADDC A, B

MOVX @R0, A

; Вычисляем 4-й байт числа

INC R0

MOV B, TMP2 + 3

MOV A, TMP3 + 2

ADDC A, B

MOVX @R0, A

; Вычисляем последний, 5-й байт числа

INC R0

MOV A, TMP3 + 3

ADDC A, #0

MOVX @R0, A

RET

; Функция осуществляет побайтовый вывод числа в UART

OUT\_UART:

; Настройка UART

MOV SCON, #0D0H

MOV PCON, #080H

MOV TMOD, #020H

MOV TH1, #0FEH

MOV TL1, #0FEH

SETB TR1

MOV P2, #00H

MOV R0, #RES

MOV R4, #06H

UART\_LOOP:

; Побайтовый вывод итогового числа

MOVX A, @R0

INC R0

MOV SBUF, A

DJNZ R4, UART\_LOOP

RET

END

## Вывод:

Все поставленные задачи были успешно выполнены! В рамках данной работы были получены навыки работы с UART контроллера. Для этого было вычислено произведение двух трехбайтных чисел и результат побайтно выведен в UART.

# Задание 4

Реализовать задержку программным способом, используя вызов подпрограммы. Частота работы МК равна 12 МГц. Точность длительности задержки должна быть выше, чем 0.01%. Заданная задержка – 2.2 сек.

Расчет погрешности:

## Алгорит:

Функция main:

НАЧАЛО.

1. Вызов программной задержки delay.

КОНЕЦ.

Функция delay:

НАЧАЛО.

1. Устанавливаем счетчик l = 16H.

2. Устанавливаем счетчик i = 64H.

3. Устанавливаем счетчик j = f9H.

4. Устанавливаем счетчик k = f8H.

5. Декриментируем k.

6. Если k != 0, то переходим в пункт 5.

7. Декриментируем j.

8. Если j != 0, то переходим в пункт 7.

9. Декриментируем i.

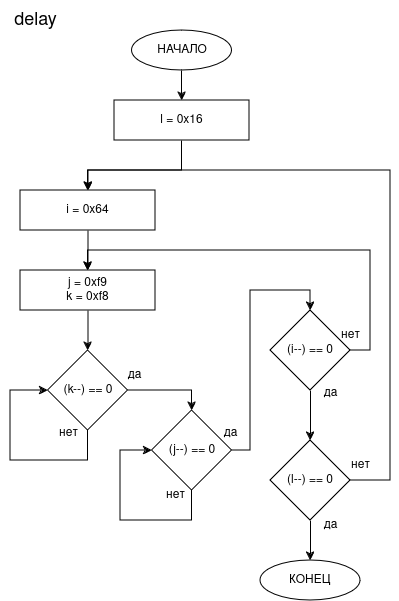
10. Если i != 0, то переходим в пункт 3.

11. Декриментируем l.

12. Если l != 0, то переходим в пункт 2.

КОНЕЦ.

## Блок схема:



## Код:

NAME DELAY

MAIN SEGMENT CODE

VARS SEGMENT DATA

RSEG VARS

DELAY\_TIME: DS 4

CSEG AT 0

LJMP START

RSEG MAIN

USING 0

DELAY:

; Устанавливаем счетчик l = 16H.

MOV DELAY\_TIME+3, #016H

DELAY\_100:

; Устанавливаем счетчик i = 64H.

MOV DELAY\_TIME+2, #064H

DELAY\_MS:

; Устанавливаем счетчик j = f9H.

MOV DELAY\_TIME, #0F9H

; Устанавливаем счетчик k = f8H.

MOV DELAY\_TIME+1, #0F8H

; Ждем пока j станет равен 0

DJNZ DELAY\_TIME, $

; Ждем пока k станет равен 0

DJNZ DELAY\_TIME+1, $

; Ждем пока i станет равен 0

DJNZ DELAY\_TIME+2, DELAY\_MS

; Ждем пока l станет равен 0

DJNZ DELAY\_TIME+3, DELAY\_100

RET

START:

; Задание начального адреса стека

MOV SP, #040H

; Вызов функции программной задержки

CALL DELAY

JMP $

END

Вывод:

Все поставленные задачи успешно выполнены. В рамках данной работы с помощью 4-х счетчиков и циклов была реализована программная задержка. Данная задержка имеет погрешность около 0.004%.

# Задание 5

Реализовать задержку аппаратным способом используя таймер T0. Заданная задержка – 7.1 сек.

Расчет погрешности:

## Алгорит:

Функция main:

НАЧАЛО.

1. Настраиваем прерывание таймера.

2. Устанавливаем время задержки.

3. Вызов функции delay.

КОНЕЦ.

Функция delay:

НАЧАЛО.

1. Настройка 0 таймера.

2. Запуск таймера.

3. Если tr1 == 1, то перейти в пункт 3.

КОНЕЦ.

Функция обработчик прерывания таймера:

НАЧАЛО.

1. Задаем th0, tl0.

2. Декрементируем R0.

3. Если R0 != 0, то переходим в п. 7.

4. Декрементируем R1.

5. Задаем R0.

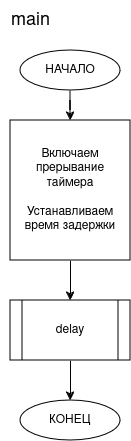
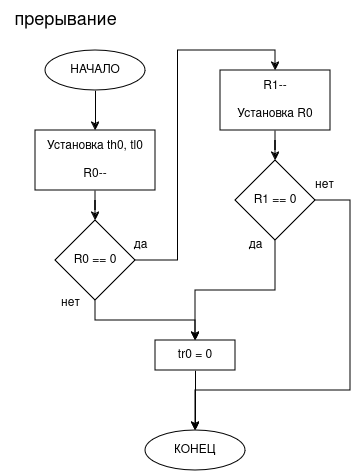
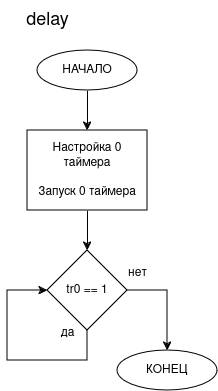
6. Если R1 != 0, то переходим в п. 7.

7. Очищаем tr0.

8. Выходим из прерывания.

КОНЕЦ.

## Блок схема:

## Код:

NAME TIMER\_DELAY

MAIN SEGMENT CODE

CSEG AT 0BH

LJMP TIMER\_INT

CSEG AT 0

LJMP START

RSEG MAIN

USING 0

START:

; Настраиваем прерывание 0 таймера

SETB IE.7

SETB IE.1

; Задаем время задержки

MOV R0, #0AH

MOV R1, #04CH

CALL TIMER\_DELAY

JMP $

TIMER\_DELAY:

; Настраиваем 0 таймер

MOV TMOD, #001H

MOV TL0, #0F6H

MOV TH0, #0D8H

; Запускаем таймер

SETB TR0

; Ждем окончание таймера

JB TR0, $

RET

TIMER\_INT:

; Задаем значение счетчика таймера

MOV TL0, #0F6H

MOV TH0, #0D8H

; Уменьшаем младший счетчик

DEC R0

MOV A, R0

JZ TIMER\_INT\_H

RETI

TIMER\_INT\_H:

; Если младший счетчик == 0, то уменьшаем старший счетчик

DEC R1

MOV R0, #0AH

MOV A, R1

; Если старший счетчик == 0, то выключаем таймер

JZ TIMER\_INT\_EXIT

RETI

TIMER\_INT\_EXIT:

; Выключаем таймер

CLR TR0

RETI

END

## Вывод:

Все поставленные задачи были успешно выполнены. В ходе данной работы была реализована задержка с помощью аппаратных средств (0 таймера), погрешность работы которой составила 0.0014%.

# Задание 6

Написать программу «Бегущие огни», которая должна выводить световые эффекты с помощью красных светодиодов. Эффекты переключаются двумя кнопками. При достижении максимального или минимального номера эффекта на 0.5с. должен загореться зеленый светодиод. Другая пара кнопок должна регулировать скорость смены кадров в эффектах. При достижении максимальной или минимальной скорости должен загораться зеленый светодиод на 1с.

## Алгорит:

Функция main:

НАЧАЛО.

1. Настройка прерываний.

2. Настройка приоритетов прерываний.

3. Настройка таймера.

4. Вызов slp20.

5. Вызов exec\_mode.

КОНЕЦ.

Функция slp20:

НАЧАЛО.

1. Устанавливаем значение переменной таймера в 20 сек.

2. Запуск 1 таймера.

КОНЕЦ.

Функция slp30:

НАЧАЛО.

1. Установка значения переменной таймера в 30 сек.

2. Запуск 1 таймера.

КОНЕЦ.

Функция delay:

НАЧАЛО.

1. Записать значение скорости в счетчик.

2. Вызов delay\_100.

3. Декремент счетчика.

4. Если счетчик != 0, то переход в п. 3.

КОНЕЦ.

Функция delay\_100:

НАЧАЛО.

1. Устанавливаем счетчик i = 64H.

2. Устанавливаем счетчик j = f9H.

3. Устанавливаем счетчик k = f8H.

4. Декриментируем k.

5. Если k != 0, то переходим в пункт 4.

6. Декриментируем j.

7. Если j != 0, то переходим в пункт 6.

8. Декриментируем i.

9. Если i != 0, то переходим в пункт 2.

КОНЕЦ.

Функция exec\_mode:

НАЧАЛО.

1. Записываем в рег. А текущий режим.

2. Записываем в порт 1 значение рег. А.

3. Смещаем циклично биты рег. А в право.

4. Вызов delay.

КОНЕЦ.

Функция select\_mode:

НАЧАЛО.

1. Вызов slp20.

2. Записываем в порт 1 значение индекса текущего режима.

3. Вызов delay\_100.

4. Если P3.4 != 0, то переход в п. 7

5. Вызов next\_mode.

6. Переход в п. 11

7. Если P3.5 != 0, то переход в п. 10

8. Вызов prev\_mode.

9. Переход в п. 11

10. Вызов exit\_int.

11. Выход из прерывания

КОНЕЦ.

Функция prev\_mode:

НАЧАЛО.

1. Присвоить рег. А значение индекса текущего режима.

2. Если рег. А == 0, то переход в п. 6

3. Вызов blip\_mode.

4. Присвоить рег. А значение 0xF.

5. Перехов в п. 7

6. Декремент значение рег. А.

7. Присвоить индексу режима значение рег. А.

8. Вызов exit\_int.

КОНЕЦ.

Функция next\_mode:

НАЧАЛО.

1. Присвоить рег. А разность индекса режима и 0xF.

2. Если рег. А == 0, то переход в п. …

3. Присвоить рег. А значение индекса режима.

4. Инкремент рег. А.

5. Переход в п. 8.

6. Вызов blip\_mode.

7. Присвоить рег. А значение 0x0.

8. Присвоить индексу режима значение рег. А.

9. Вызов exit\_int.

КОНЕЦ.

Функция blip\_mode:

НАЧАЛО.

1. Присвоить AR0 = 0xA

2. Присвоить AR1 = 0x5

3. Присвоить AR2 = 0x3

4. Присвоить P3.7 = 0x0

5. Запуск 0 таймера.

КОНЕЦ.

Функция blip\_spd:

НАЧАЛО.

1. Присвоить AR0 = 0xA

2. Присвоить AR1 = 0x5

3. Присвоить P3.7 = 0x0

4. Запуск 0 таймера.

КОНЕЦ.

Функция select\_spd:

НАЧАЛО.

1. Вызов slp20.

2. Записываем в порт 1 значение индекса текущей скорости.

3. Вызов delay\_100.

4. Если P3.4 != 0, то переход в п. 7

5. Вызов next\_spd.

6. Переход в п. 11

7. Если P3.5 != 0, то переход в п. 10

8. Вызов prev\_spd.

9. Переход в п. 11

10. Вызов exit\_int.

11. Выход из прерывания

КОНЕЦ.

Функция prev\_spd:

НАЧАЛО.

1. Присвоить рег. А значение индекса текущей скорости.

2. Если рег. А == 0, то переход в п. 6

3. Вызов blip\_spd.

4. Присвоить рег. А значение 0xF.

5. Перехов в п. 7

6. Декремент значение рег. А.

7. Присвоить индексу скорости значение рег. А.

8. Вызов exit\_int.

КОНЕЦ.

Функция next\_spd:

НАЧАЛО.

1. Присвоить рег. А разность индекса скорости и 0xF.

2. Если рег. А == 0, то переход в п. …

3. Присвоить рег. А значение индекса скорости.

4. Инкремент рег. А.

5. Переход в п. 8

6. Вызов blip\_spd.

7. Присвоить рег. А значение 0x0.

8. Присвоить индексу скорости значение рег. А.

9. Вызов exit\_int.

КОНЕЦ.

Функция timer0:

НАЧАЛО.

1. Присваиваем tl0 = 0xf6

2. Присваиваем tr0 = 0xd8

3. Декремент AR0

4. Присваиваем рег. А значение рег. AR0

5. Если рег. А != 0, то переход в п. 25

6. Декремент AR1

7. Присваиваем рег. AR0 = 0xA

8. Присваиваем рег. А значение рег. AR1

9. Если рег. А != 0, то переход в п. 25

10. Присвоить рег. А разность значения рег. AR7 и 0x3

11. Если рег. А == 0, то переход в п. 22

12. Присвоить рег. А разность значения рег. AR7 и 0xf

13. Если рег. А == 0, то переход в п. 18

14. Присвоить AR7 = 0x0

15. Присвоить P3.7 = 1

16. Установить флаг tr0 = 0

17. Переход в п. 25

18. Присвоить P3.7 = 0

19. Присвоить AR7 = 0x0

20. Присвоить AR1 = 0x5

21. Переход в п. 25

22. Присвоить P3.7 = 0

23. Присвоить AR7 = 0x1

24. Присвоить AR1 = 0x5

25. Выход из прерывания

КОНЕЦ.

Функция timer1:

НАЧАЛО.

1. Присваиваем tl1 = 0xf6

2. Присваиваем tr1 = 0xd8

3. Декремент переменной sleep\_time[0]

4. Присвоить рег. А = sleep\_time[0]

5. Если рег. А == 0, то переход в п. 17

6. Присвоить sleep\_time[0] = 0xA

7. Декремент переменной sleep\_time[1]

8. Присвоить рег. А = sleep\_time[1]

9. Если рег. А == 0, то переход в п. 17

10. Присвоить sleep\_time[1] = 0x1

11. Декремент переменной sleep\_time[2]

12. Присвоить рег. А = sleep\_time[2]

13. Если рег. А == 0, то переход в п. 17

14. Установить флаг tr1 = 0

15. Вызов slp30

16. Вызов next\_mode

17. Выход из прерывания

КОНЕЦ.

## Блок схема:

## 

## 

## Код:

NAME LIGHTS

MAIN SEGMENT CODE

LITTERALS SEGMENT CODE

INTERUPTS SEGMENT CODE

; Создание переменной индекса текущего режима

MOD EQU 038H

; Создание переменной индекса текущей скорости

SPD EQU 039H

DSEG AT 030H

DELAY\_TIME: DS 4

SLEEP\_TIME: DS 3

; Точка входа программы

CSEG AT 0

LJMP MAIN

; Назначение точек входов в различные прерывания

CSEG AT 03H

LJMP SELECT\_MODE

CSEG AT 013H

LJMP SELECT\_SPD

CSEG AT 0BH

LJMP TIMER0

CSEG AT 01BH

LJMP TIMER1

RSEG LITTERALS

; Список режимов

MODS: DB 0FEH, 0FCH, 0F8H, 0F0H, 0E0H, 0C0H, 080H, 0FAH, 0EAH, 0AAH, 0E4H, 0A4H, 099H, 011H, 0B3H, 0BDH

; Список скоростей

SPDS: DB 002H, 004H, 006H, 008H, 00AH, 00CH, 00EH, 010H, 012H, 014H, 016H, 018H, 01AH, 01CH, 01EH, 020H

RSEG INTERUPTS

USING 1

; Функция обрабатывает прерывание нажатия кнопки, и осуществляет выполнение меню выбора режима

SELECT\_MODE:

CALL SLP20

MOV A, MOD

XRL A, #0FFH

MOV P1, A

CALL DELAY\_100

JNB P3.4, NEXT\_MODE

JNB P3.5, PREV\_MODE

LJMP EXIT\_INT

; Функция выбора следующего режима

NEXT\_MODE:

CLR C

MOV A, MOD

SUBB A, #0FH

JZ NEXT\_MODE\_RESET

MOV A, MOD

INC A

LJMP MODE\_EXIT

NEXT\_MODE\_RESET:

CALL BLIP\_MOD

MOV A, #00H

LJMP MODE\_EXIT

; Функция выбора предыдущего режима

PREV\_MODE:

CLR C

MOV A, MOD

JZ PREV\_MODE\_RESET

DEC A

LJMP MODE\_EXIT

PREV\_MODE\_RESET:

CALL BLIP\_MOD

MOV A, #0FH

LJMP MODE\_EXIT

; метка осуществляет завершение выбора режима

MODE\_EXIT:

MOV MOD, A

LJMP EXIT\_INT

; Функция служит для мигания диодом при достижении крайних режимов

BLIP\_MOD:

MOV AR0, #0AH

MOV AR1, #05H

MOV AR7, #000000011B

CLR P3.7

SETB TR0

RET

; Функция обрабатывает прерывание нажатия кнопки, и осуществляет выполнение меню выбора скорости

SELECT\_SPD:

CALL SLP20

MOV A, SPD

XRL A, #0FFH

MOV P1, A

CALL DELAY\_100

JNB P3.4, NEXT\_SPD

JNB P3.5, PREV\_SPD

LJMP EXIT\_INT

; Функция выбора следующей скорости

NEXT\_SPD:

CLR C

MOV A, SPD

SUBB A, #0FH

JZ NEXT\_SPD\_RESET

MOV A, SPD

INC A

LJMP SPD\_EXIT

NEXT\_SPD\_RESET:

CALL BLIP\_SPD

MOV A, #00H

LJMP SPD\_EXIT

; Функция выбора предыдущей скорости

PREV\_SPD:

CLR C

MOV A, SPD

JZ PREV\_SPD\_RESET

DEC A

LJMP SPD\_EXIT

PREV\_SPD\_RESET:

CALL BLIP\_SPD

MOV A, #0FH

LJMP SPD\_EXIT

; Метка выхода для функция выбора скоростей

SPD\_EXIT:

MOV SPD, A

LJMP EXIT\_INT

; Функция служит для мигания диодом при достижении крайних скоростей

BLIP\_SPD:

MOV AR0, #0AH

MOV AR1, #0AH

CLR P3.7

SETB TR0

RET

; Функция, служащая для корректного выхода из прерываний для выбирающих скорости или режимы рутин.

EXIT\_INT:

MOV DPTR, #EXEC\_MODE

MOV A, DPL

PUSH ACC

MOV A, DPH

PUSH ACC

RETI

; Функция обработки прерывания 0 таймера

TIMER0:

MOV TL0, #0F6H

MOV TH0, #0D8H

DEC AR0

MOV A, AR0

JZ TIMER0\_0

RETI

TIMER0\_0:

DEC AR1

MOV AR0, #0AH

MOV A, AR1

JZ TIMER0\_1

RETI

TIMER0\_1:

CLR C

MOV A, AR7

SUBB A, #011B

JZ TIMER0\_2

CLR C

MOV A, AR7

SUBB A, #01B

JZ TIMER0\_3

JMP TIMER0\_DONE

TIMER0\_2:

SETB P3.7

MOV AR1, #05H

MOV AR7,#01B

RETI

TIMER0\_3:

CLR P3.7

MOV AR1, #05H

MOV AR7, #00B

RETI

TIMER0\_DONE:

MOV AR7, #00H

SETB P3.7

CLR TR0

RETI

; Функция обработки прерывания 1 таймера

TIMER1:

MOV TL1, #0F6H

MOV TH1, #0D8H

DEC SLEEP\_TIME

MOV A, SLEEP\_TIME

JZ TIMER1\_1

RETI

TIMER1\_1:

MOV SLEEP\_TIME, #0AH

DEC SLEEP\_TIME+1

MOV A, SLEEP\_TIME+1

JZ TIMER1\_2

RETI

TIMER1\_2:

MOV SLEEP\_TIME+1, #01H

DEC SLEEP\_TIME+2

MOV A, SLEEP\_TIME+2

JZ TIMER1\_DONE

RETI

TIMER1\_DONE:

CLR TR1

LCALL SLP30

LJMP NEXT\_MODE

RSEG MAIN

USING 0

MAIN: ; Инициализация основных параметров и компонентов программы

MOV IE, #010001111B

MOV IP, #0AH

MOV TMOD, #011H

MOV MOD, #00H

MOV SPD, #00H

MOV SP, #040H

CALL SLP20

LJMP EXEC\_MODE

; Основная функция, в рамках которой происходит выполнения определенного режима с выбранной скоростью

EXEC\_MODE:

MOV SP, #040H

MOV A, MOD

MOV DPTR, #MODS

MOVC A, @A+DPTR

EXEC\_MODE\_LOOP:

MOV P1, A

RL A

PUSH ACC

CALL DELAY

POP ACC

JMP EXEC\_MODE\_LOOP

; Функция создания программной задержки

DELAY:

MOV A, SPD

MOV DPTR, #SPDS

MOVC A, @A+DPTR

MOV DELAY\_TIME+3, A

DELAY\_LOOP:

CALL DELAY\_100

DJNZ DELAY\_TIME+3, DELAY\_LOOP

RET

; Функция для создания задержки в 100 милисекунд

DELAY\_100:

MOV DELAY\_TIME+2, #064H

DELAY\_MS:

MOV DELAY\_TIME, #0F9H

MOV DELAY\_TIME+1, #0F8H

DJNZ DELAY\_TIME, $

DJNZ DELAY\_TIME+1, $

DJNZ DELAY\_TIME+2, DELAY\_MS

RET

; Функция для задания таймеру 1 режима ожидания в 20 секунд

SLP20:

MOV SLEEP\_TIME, #0AH

MOV SLEEP\_TIME+1, #0C8H

MOV SLEEP\_TIME+2, #01H

SETB TR1

RET

; Функция для задания таймеру 1 режима ожидания в 30 секунд

SLP30:

MOV SLEEP\_TIME, #0AH

MOV SLEEP\_TIME+1, #0FFH

MOV SLEEP\_TIME+2, #02DH

SETB TR1

RET

END

## Вывод:

Все поставленные задачи успешно выполнены. В ходе данной работы было разработано приложения “бегающие огни”, включающее в себя возможность переключать различные режимы подсветки и скорости ее изменения с помощью функционального меню, имеющего отображение индекс текущего режима или скорости.

# Вывод

Все поставленные задачи успешно выполнены. В ходе работы были: изучена архитектура и организация микроконтроллеров семейства Intel 8051; изучены основы программирования микропроцессорных систем на базе микроконтроллера Atmel AT89C51; написаны программ согласно заданиям, работа из 6 программы была проверена на макетной плате.