МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

Лабораторная работа № 1

Изучение микроконтроллеров семейства Intel 8051.

Выполнил студент 595 гр.

_____ А.В. Лаптев

Проверил:

_____ В.В. Белозерских

Лабораторная работа защищена

«___» _____2023 г.

Оценка _____

Барнаул 2023

Цель работы: Изучение архитектуры и организации микроконтроллеров семейства Intel 8051. Получение навыков программирования и тестирования и отладки устройств на базе микроконтроллеров семейства MCS-51 с использованием среды разработки ProView32 и программатора ProAtMic.

Задачи: Изучить архитектуру и организацию микроконтроллеров семейства Intel 8051. Изучить основы программирования микропроцессорных систем на базе микроконтроллера Atmel AT89C51. В соответствии с заданием написать и отладить программы с помощью среды программирования ProView32 и произвести программирование контроллера с помощью программатора ProAtMic. Работоспособность созданного устройства проверить на лабораторной установке.

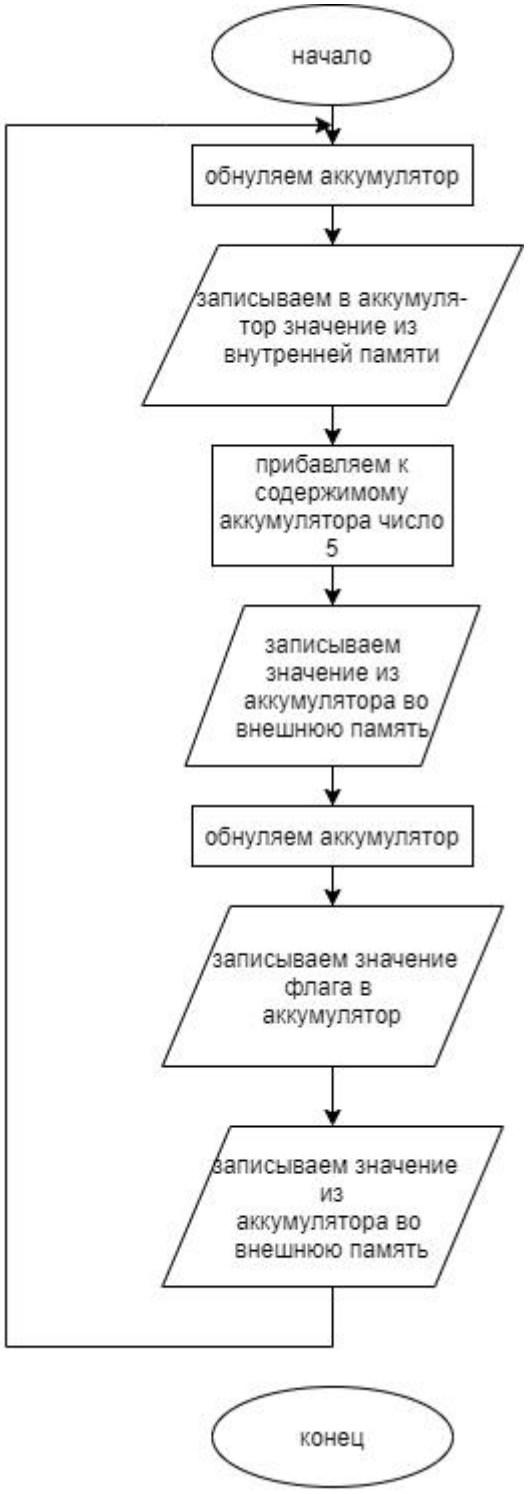
Ход работы:

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

Алгоритм:

1. Начало;
2. Значение из регистра, находящегося во второй половине внутренней памяти данных, заносим в аккумулятор;
3. Складываем содержимое аккумулятора с числом 5;
4. Адресуем ячейку из внешней памяти данных;
5. Заносим во внешнюю память данных содержимое аккумулятора;
6. Заносим значение флага переноса в аккумулятор;
7. Адресуем ячейку из внешней памяти данных;
8. Заносим во внешнюю память данных содержимое аккумулятора;
9. Конец.

Блок-схема алгоритма:



Текст программы

```
NAME LAB1
MAIN SEGMENT CODE
CSEG AT 0
LJMP start
USING 0
RSEG MAIN
start:
MOV A, #5                ;ЗАНОСИМ ЧИСЛО 5 В АККУМУЛЯТОР
ADD A, 78H               ;СКЛАДЫВАЕМ С ЧИСЛОМ, КОТОРОЕ НАХОДИТЬСЯ ПО ЭТОМУ АДРЕСУ
MOV DPTR, #1
MOVX @DPTR, A            ;ЗАНОСИМ РЕЗУЛЬТАТ ВО ВНЕШНЮЮ ПАМЯТЬ
CLR A
ADDC A, #0               ;ПОЛУЧАЕМ ЗНАЧЕНИЕ ПЕРЕНОСА
MOV DPTR, #0
MOVX @DPTR, A            ;ЗАНОСИМ РЕЗУЛЬТАТ ВО ВНЕШНЮЮ ПАМЯТЬ
LJMP start
END
```

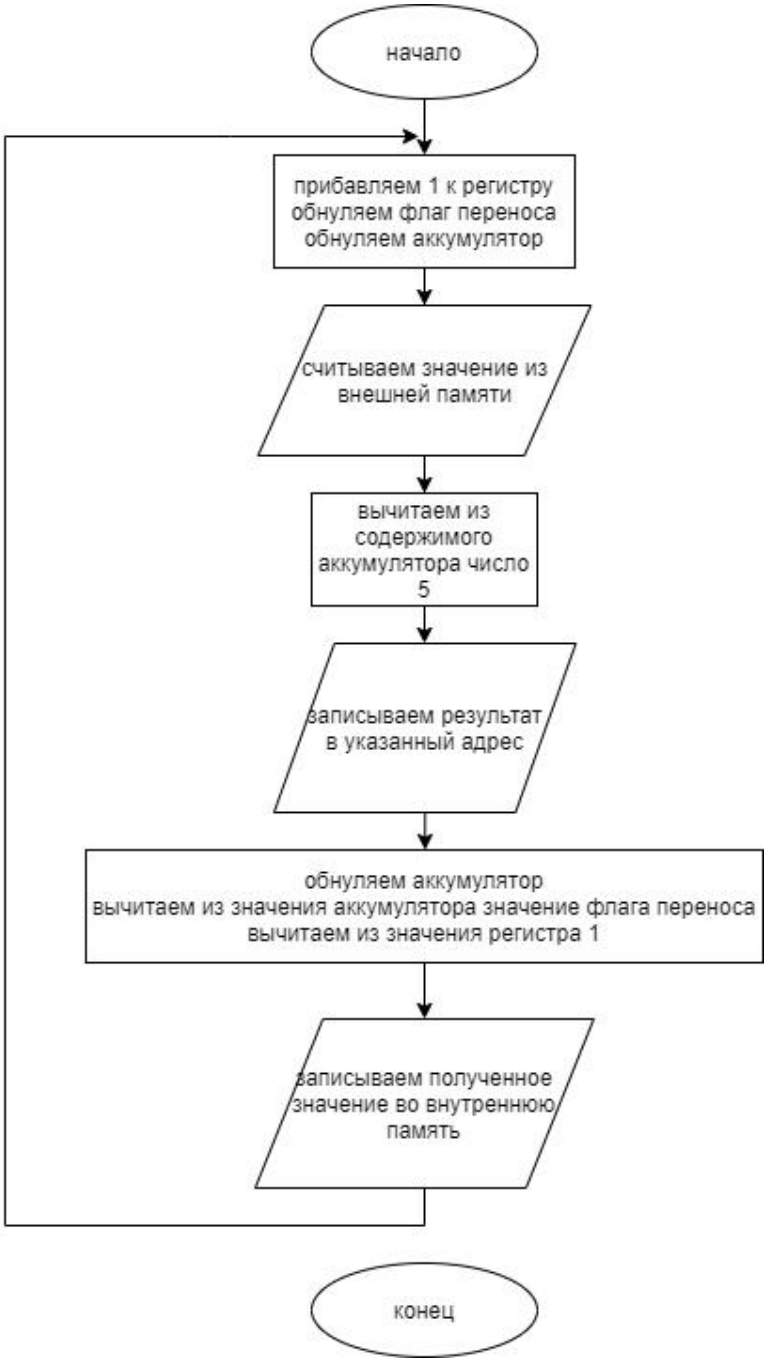
Вывод: Задание выполнено успешно. В ходе выполнения работы были получены навыки работы с внешней и внутренней памятью МК и написана программа сложения числа, находящегося во второй половине внутренней памяти, с числом 5. Результат программы помещается во внешнюю память.

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

Алгоритм:

1. Начало;
2. Прибавляем 1 к регистру;
3. Обнуляем аккумулятор;
4. Обнуляем флаг переноса;
5. Адресовать ячейку из внешней памяти данных;
6. Заносим в аккумулятор содержимое из внешней памяти данных;
7. Вычитаем из содержимого аккумулятора число 5;
8. Заносим значение в указанный адрес;
9. Обнуляем аккумулятор;
10. Вычитаем из значения аккумулятора значение флага переноса;
11. Вычитаем из значения регистра 1;
12. Записываем полученное значение во внутреннюю память;
13. Конец.

Блок-схема алгоритма:



Текст программы

```
NAME LAB1

MAIN SEGMENT CODE

CSEG AT 0

LJMP start

USING 0

RSEG MAIN

start:

MOV DPTR, #0

MOVX A, @DPTR ;ЗАГРУЖАЕМ ЧИСЛО ИЗ ВНЕШНЕЙ ПАМЯТИ В АККУМУЛЯТОР

SUBB A, #5 ;ВЫЧИТАЕМ ИЗ ЗНАЧЕНИЯ В АККУМУЛЯТОРЕ 5

MOV R0, #01H

MOV @R0, A ;ЗАНОСИМ РЕЗУЛЬТАТ ВО ВНУТРЕННЮЮ ПАМЯТЬ, ИСПОЛЬЗУЯ КОСВЕННУЮ АДРЕСАЦИЮ

CLR A

ADDC A, #0 ;ПОЛУЧАЕМ ЗНАЧЕНИЕ ПЕРЕНОСА

MOV R0, #00H

MOV @R0, A ;ЗАНОСИМ РЕЗУЛЬТАТ ВО ВНУТРЕННЮЮ ПАМЯТЬ, ИСПОЛЬЗУЯ КОСВЕННУЮ АДРЕСАЦИЮ

LJMP start

END
```

Вывод: Задание выполнено успешно. В ходе выполнения работы были получены навыки работы с памятью с использованием косвенной адресации и написана программа вычитания из числа, находящегося во внешней памяти данных МК. Результат программы помещается во внутреннюю память данных, используя косвенную адресацию.

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

Алгоритм основной программы:

1. Начало;
2. Загружаем данные из внешней памяти;
3. Загружаем 3 байта из порта 2, от старшего к младшему в регистры R5, R6, R7;
4. Переносим байты первого числа из внешней памяти в регистры R1, R2, R3;
5. Заносим значение R7 в R4 для временного хранения для умножения;
6. Вызов подпрограммы умножения;
7. Переносим результаты умножения из временных ячеек;
8. Заносим значение R6 в R4 для временного хранения для умножения;
9. Вызов подпрограммы умножения;
10. Переносим результаты умножения из временных ячеек;
11. Заносим значение R5 в R4 для временного хранения для умножения;
12. Вызов подпрограммы умножения;
13. Переносим результаты умножения из временных ячеек;
14. Записываем результат первого и второго умножения в регистры R1, R2, R3, R4, R5, R6, R7;
15. Записываем младший байт конечного результата;
16. Вызываем подпрограмму сложения;

17. Записываем следующий байт конечного результата;
18. Записываем результат суммирования и третьего умножения в регистры R1, R2, R3, R4, R5, R6, R7;
19. Вызываем подпрограмму сложения;
20. Записываем байты конечного результата;
21. Настраиваем и включаем UART;
22. Последовательно начиная со старшего байта выводим результат в UART;
23. Конец.

Алгоритм подпрограммы умножения:

1. Начало;
2. Умножаем младший байт первого числа на байт второго и записываем результат во временную ячейку;
3. Умножаем средний байт первого числа на байт второго и записываем результат во временную ячейку;
4. Умножаем старший байт первого числа на байт второго и записываем результат во временную ячейку
5. Конец.

Алгоритм подпрограммы сложения:

1. Начало;
2. Складываем младшие байты и заносим в конечную переменную;
3. Складываем средние байты и заносим в конечную переменную;
4. Складываем следующий байт и заносим в конечную переменную;
5. Складываем старший байт второго числа с переносом от предыдущего сложения и заносим в конечную переменную;

6. Конец.

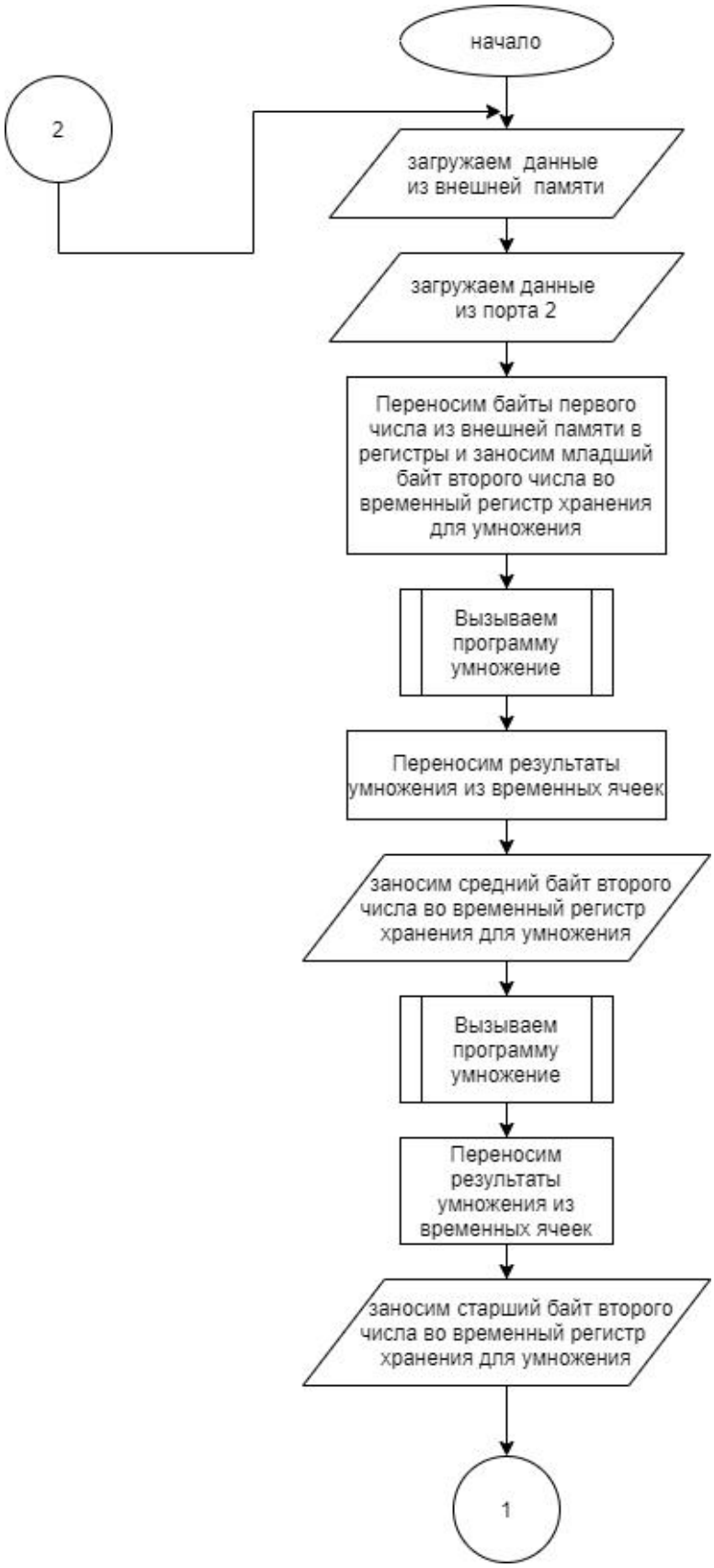
Алгоритм подпрограммы вывода:

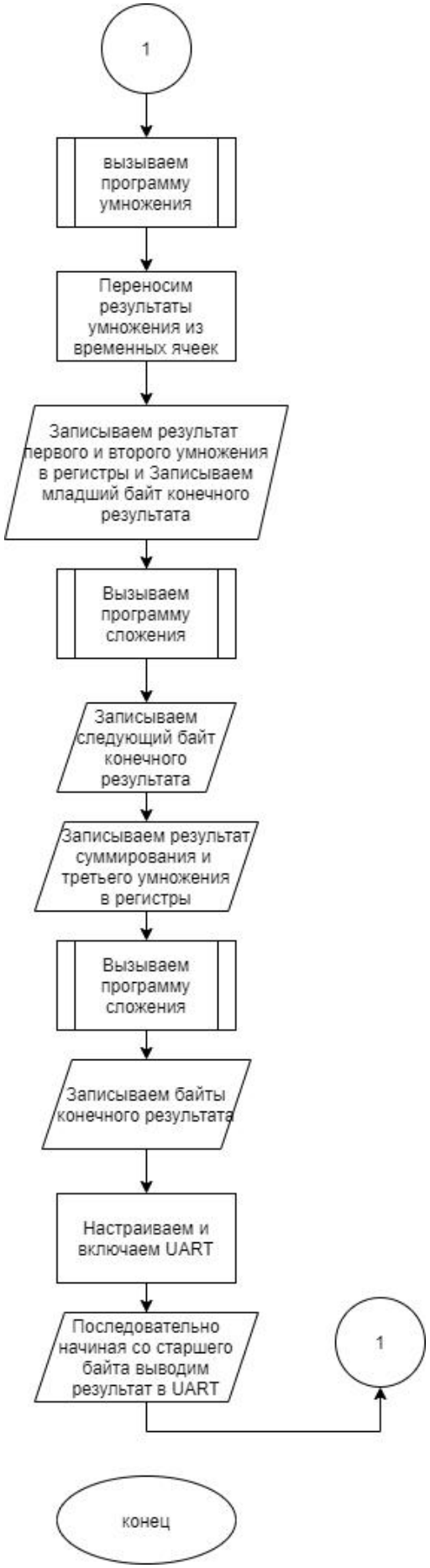
1. Начало;
2. Ожидание пока появится флаг прерывания передатчика;
3. Очищаем флаг прерывания передатчика;
4. Отправляем байт в буфер передачи;
5. Конец.

Инициализация всех символический переменных происходит в отдельном файле.

Блок-схема алгоритма:

Основная программа:




1
вызываем программу умножения
Переносим результаты умножения из временных ячеек
Записываем результат первого и второго умножения в регистры и Записываем младший байт конечного результата
Вызываем программу сложения
Записываем следующий байт конечного результата
Записываем результат суммирования и третьего умножения в регистры
Вызываем программу сложения
Записываем байты конечного результата
Настраиваем и включаем UART
Последовательно начиная со старшего байта выводим результат в UART
1
конец

Подпрограмма SUMMATION:

Подпрограмма OUTPUT_UART:

Подпрограмма MULTIPLICATION:

Текст программы

Основная программа:

```
NAME LAB_1

EXTERN CODE (LAB_1, OUTPUT, MULT, PARAMS, SUMM)

EXTERN data (A_BYTE_1, A_BYTE_2, A_BYTE_3, B_BYTE_1, B_BYTE_2,
B_BYTE_3, FIRST_MULT_RESULT_1, FIRST_MULT_RESULT_2,
FIRST_MULT_RESULT_3, SECOND_MULT_RESULT_1,
SECOND_MULT_RESULT_2, SECOND_MULT_RESULT_3,
THIRD_MULT_RESULT_1, THIRD_MULT_RESULT_2,
THIRD_MULT_RESULT_3, FIRST_BYTE_ADDRESS,
SECOND_BYTE_ADDRESS, THIRD_BYTE_ADDRESS, SUMM_1, SUMM_2,
SUMM_3, SUMM_4, SUMM_5, SUMM_6)

MAIN SEGMENT CODE

CSEG AT 0

PROG SEGMENT    CODE

CONST       SEGMENT    CODE

STACK       SEGMENT    IDATA

RSEG   STACK

        DS      10H

        CSEG   AT     0

        USING         0

JMP start

start:

; СЧИТЫВАЕМ ДАННЫЕ ИЗ ВНЕШНЕЙ ПАМЯТИ

MOV DPTR, #A_BYTE_1

MOVX A, @DPTR

MOV R1, A

MOV DPTR, #A_BYTE_2

MOVX A, @DPTR

MOV R2, A

MOV DPTR, #A_BYTE_3

MOVX A, @DPTR
```

```
MOV R3, A
; СЧИТЫВАЕМ ВТОРОЕ ЧИСЛО ПОБАЙТНО ИЗ ПОРТА P2
MOV R4, P2
MOV R5, P2
MOV R6, P2
; ПЕРЕНОСИМ МЛАДШИЕ БАЙТЫ ДЛЯ УМНОЖЕНИЯ
MOV A, R6
MOV R7, A
; ВЫЗЫВАЕМ ПОДПРОГРАММУ ДЛЯ УМНОЖЕНИЯ ОПЕРАНДОВ
CALL MULT
MOV R0, #FIRST_BYTE_ADDRESS
MOV A, @R0
MOV R0, #FIRST_MULT_RESULT_1
MOV @R0, A
MOV R0, #SECOND_BYTE_ADDRESS
MOV A, @R0
MOV R0, #FIRST_MULT_RESULT_2
MOV @R0, A
MOV R0, #THIRD_BYTE_ADDRESS
MOV A, @R0
MOV R0, #FIRST_MULT_RESULT_3
MOV @R0, A
MOV R0, #FIRST_BYTE_ADDRESS - 1
MOV A, @R0
MOV R0, #FIRST_MULT_RESULT_1 - 1
MOV @R0, A
; ПЕРЕНОСИМ СРЕДНИЕ БАЙТЫ ДЛЯ УМНОЖЕНИЯ
MOV A, R5
MOV R7, A
```

```
; ВЫЗЫВАЕМ ПОДПРОГРАММУ ДЛЯ УМНОЖЕНИЯ ОПЕРАНДОВ
CALL MULT
MOV R0, #FIRST_BYTE_ADDRESS
MOV A, @R0
MOV R0, #SECOND_MULT_RESULT_1
MOV @R0, A
MOV R0, #SECOND_BYTE_ADDRESS
MOV A, @R0
MOV R0, #SECOND_MULT_RESULT_2
MOV @R0, A
MOV R0, #THIRD_BYTE_ADDRESS
MOV A, @R0
MOV R0, #SECOND_MULT_RESULT_3
MOV @R0, A
MOV R0, #FIRST_BYTE_ADDRESS - 1
MOV A, @R0
MOV R0, #SECOND_MULT_RESULT_1 - 1
MOV @R0, A
; ПЕРЕНОСИМ СТАРШИЕ БАЙТЫ ДЛЯ УМНОЖЕНИЯ
MOV A, R4
MOV R7, A
; ВЫЗЫВАЕМ ПОДПРОГРАММУ ДЛЯ УМНОЖЕНИЯ ОПЕРАНДОВ
CALL MULT
MOV R0, #FIRST_BYTE_ADDRESS
MOV A, @R0
MOV R0, #THIRD_MULT_RESULT_1
MOV @R0, A
MOV R0, #SECOND_BYTE_ADDRESS
MOV A, @R0
```

```
MOV R0, #THIRD_MULT_RESULT_2
MOV @R0, A
MOV R0, #THIRD_BYTE_ADDRESS
MOV A, @R0
MOV R0, #THIRD_MULT_RESULT_3
MOV @R0, A
MOV R0, #FIRST_BYTE_ADDRESS - 1
MOV A, @R0
MOV R0, #THIRD_MULT_RESULT_1 - 1
MOV @R0, A
;СУММИРУЕМ СТОЛБИКИ
CALL SUMM
;UART
MOV TMOD, #00100000B
MOV TH1, #0FDH
SETB TR1
MOV PCON, #11011100B
MOV SCON, #01010010B
MOV R0, #SUMM_1
MOV A, @R0
CALL OUTPUT
MOV R0, #SUMM_2
MOV A, @R0
CALL OUTPUT
MOV R0, #SUMM_3
MOV A, @R0
CALL OUTPUT
MOV R0, #SUMM_4
MOV A, @R0
```

```
CALL OUTPUT

MOV R0, #SUMM_5

MOV A, @R0

CALL OUTPUT

MOV R0, #SUMM_6

MOV A, @R0

CALL OUTPUT

LJMP start

END
```

Подпрограмма SUMMATION:

```
NAME SUMMATION

PUBLIC SUMM

EXTERN data (FIRST_MULT_RESULT_1, FIRST_MULT_RESULT_2, FIRST_MULT_RESULT_3, SECOND_MULT_RESULT_1, SECOND_MULT_RESULT_2, SECOND_MULT_RESULT_3, THIRD_MULT_RESULT_1, THIRD_MULT_RESULT_2, THIRD_MULT_RESULT_3, SUMM_1, SUMM_2, SUMM_3, SUMM_4, SUMM_5, SUMM_6)

SUMMATION_ROUTINES SEGMENT CODE

RSEG SUMMATION_ROUTINES

JMP summ

summ:

;СКЛАДЫВАЕМ МЛАДШИЙ РАЗРЯД ЧИСЛА

MOV R0, #FIRST_MULT_RESULT_3

MOV A, @R0

MOV R0, #SUMM_6

MOV @R0, A

;СКЛАДЫВАЕМ ВТОРОЙ РАЗРЯД ЧИСЛА

MOV R0, #FIRST_MULT_RESULT_2

MOV A, @R0

MOV R0, #SECOND_MULT_RESULT_3
```

```
ADDC A, @R0
MOV R0, #SUMM_5
MOV @R0, A
MOV A, #0
ADDC A, #0
MOV R0, #SUMM_4
MOV @R0, A
;СКЛАДЫВАЕМ ТРЕТИЙ РАЗРЯД ЧИСЛА
MOV R0, #FIRST_MULT_RESULT_1
MOV A, @R0
MOV R0, #SECOND_MULT_RESULT_2
ADDC A, @R0
MOV B, A
MOV A, #0
ADDC A, #0
MOV R0, #SUMM_3
MOV @R0, A
MOV A, B
MOV R0, #THIRD_MULT_RESULT_3
ADDC A, @R0
MOV R0, #SUMM_4
ADD A, @R0
MOV @R0, A
MOV A, #0
ADDC A, #0
MOV R0, #SUMM_3
ADD A, @R0
MOV @R0, A
;СКЛАДЫВАЕМ ЧЕТВЕРТЫЙ РАЗРЯД ЧИСЛА
```

```
MOV R0, #FIRST_MULT_RESULT_1 - 1
MOV A, @R0
MOV R0, #SECOND_MULT_RESULT_1
ADDC A, @R0
MOV B, A
MOV A, #0
ADDC A, #0
MOV R0, #SUMM_2
MOV @R0, A
MOV A, B
MOV R0, #THIRD_MULT_RESULT_2
ADDC A, @R0
MOV B, A
MOV A, #0
ADDC A, #0
MOV R0, #SUMM_2
ADD A, @R0
MOV @R0, A
MOV A, B
MOV R0, #SUMM_3
ADD A, @R0
MOV @R0, A
;СКЛАДЫВАЕМ ПЯТЫЙ РАЗРЯД ЧИСЛА
MOV R0, #SECOND_MULT_RESULT_1 - 1
MOV A, @R0
MOV R0, #THIRD_MULT_RESULT_1
ADDC A, @R0
MOV B, A
MOV A, #0
```

```
ADDC A, #0
MOV R0, #SUMM_1
MOV @R0, A
MOV A, B
MOV R0, #SUMM_2
ADD A, @R0
MOV @R0, A
;СКЛАДЫВАЕМ ШЕСТОЙ РАЗРЯД ЧИСЛА
MOV R0, #THIRD_MULT_RESULT_1 - 1
MOV A, @R0
MOV R0, #SUMM_1
ADD A, @R0
MOV @R0, A
RET
END
```

Подпрограмма MULTIPLICATION:

```
NAME MULTIPLICATION
PUBLIC MULT
EXTERN data (FIRST_BYTE_ADDRESS, SECOND_BYTE_ADDRESS, THIRD_BYTE_ADDRESS)
BITVAR SEGMENT data
RSEG BITVAR
MULTIPLICATION_ROUTINES SEGMENT CODE
RSEG MULTIPLICATION_ROUTINES
JMP mult
mult:
; ПОМЕЩАЕМ ОПЕРАНДЫ В АККУМУЛЯТОР И РЕГИСТР В
MOV A, R3
MOV B, R7
MUL AB
```

```
; СОХРАНЯЕМ РЕЗУЛЬТАТ ВЫЧИСЛЕНИЙ
MOV R0, #THIRD_BYTE_ADDRESS
MOV @R0, A
MOV A, B
MOV R0, #SECOND_BYTE_ADDRESS
MOV @R0, A
; ПОМЕЩАЕМ ОПЕРАНДЫ В АККУМУЛЯТОР И РЕГИСТР B
MOV A, R2
MOV B, R7
MUL AB
; СОХРАНЯЕМ РЕЗУЛЬТАТ ВЫЧИСЛЕНИЙ
MOV R0, #SECOND_BYTE_ADDRESS
ADDC A, @R0
MOV @R0, A
MOV A, B
ADDC A, #0
MOV R0, #FIRST_BYTE_ADDRESS
MOV @R0, A
; ПОМЕЩАЕМ ОПЕРАНДЫ В АККУМУЛЯТОР И РЕГИСТР B
MOV A, R1
MOV B, R7
MUL AB
; СОХРАНЯЕМ РЕЗУЛЬТАТ ВЫЧИСЛЕНИЙ
MOV R0, #FIRST_BYTE_ADDRESS
ADDC A, @R0
MOV @R0, A
MOV A, B
ADDC A, #0
MOV R0, #FIRST_BYTE_ADDRESS - 1
```

```
MOV @R0, A
RET
END
```

Подпрограмма OUTPUT_UART:

```
NAME OUTPUT_UART
PUBLIC OUTPUT
OUTPUT_UART_ROUTINES SEGMENT CODE
RSEG OUTPUT_UART_ROUTINES
JMP output
output:
; ЖДЕМ, ПОКА ПОЯВИТСЯ ФЛАГ ПРЕРЫВАНИЯ ПЕРЕДАТЧИКА
JNB TI, $
; ОЧИЩАЕМ ФЛАГ ПРЕРЫВАНИЯ ПЕРЕДАТЧИКА
CLR TI
; ОТПРАВЛЯЕМ БАЙТ В БУФЕР ПЕРЕДАЧИ
MOV SBUF, A
MOV A, #0
RET
END
```

Подпрограмма PARAMS:

```
 NAME PARAMS
PUBLIC A_BYTE_1, A_BYTE_2, A_BYTE_3, FIRST_BYTE_ADDRESS,
SECOND_BYTE_ADDRESS, THIRD_BYTE_ADDRESS,
FIRST_MULT_RESULT_1, FIRST_MULT_RESULT_2,
FIRST_MULT_RESULT_3, SECOND_MULT_RESULT_1,
SECOND_MULT_RESULT_2, SECOND_MULT_RESULT_3,
THIRD_MULT_RESULT_1, THIRD_MULT_RESULT_2,
THIRD_MULT_RESULT_3, SUMM_1, SUMM_2, SUMM_3, SUMM_4, SUMM_5,
SUMM_6
BITVAR SEGMENT data
RSEG BITVAR
A_BYTE_1 EQU 00H
```

```
A_BYTE_2 EQU A_BYTE_1 + 1
A_BYTE_3 EQU A_BYTE_2 + 1
FIRST_BYTE_ADDRESS EQU 0DH
SECOND_BYTE_ADDRESS EQU FIRST_BYTE_ADDRESS + 1
THIRD_BYTE_ADDRESS EQU SECOND_BYTE_ADDRESS + 1
FIRST_MULT_RESULT_1 EQU 15H
FIRST_MULT_RESULT_2 EQU FIRST_MULT_RESULT_1 + 1
FIRST_MULT_RESULT_3 EQU FIRST_MULT_RESULT_2 + 1
SECOND_MULT_RESULT_1 EQU 1CH
SECOND_MULT_RESULT_2 EQU SECOND_MULT_RESULT_1 + 1
SECOND_MULT_RESULT_3 EQU SECOND_MULT_RESULT_2 + 1
THIRD_MULT_RESULT_1 EQU 23H
THIRD_MULT_RESULT_2 EQU THIRD_MULT_RESULT_1 + 1
THIRD_MULT_RESULT_3 EQU THIRD_MULT_RESULT_2 + 1
SUMM_1 EQU 2AH
SUMM_2 EQU SUMM_1 + 1
SUMM_3 EQU SUMM_2 + 1
SUMM_4 EQU SUMM_3 + 1
SUMM_5 EQU SUMM_4 + 1
SUMM_6 EQU SUMM_5 + 1
END
```

Вывод: Задание выполнено успешно. В ходе выполнения работы были изучены основы работы с UART и освоено функциональное разделение программного кода на части и реализована программа умножения двух трехбайтных чисел. Первое число находится во внешней памяти, второе число принимается побайтно из порта 2 микроконтроллера. Результата умножения выводится в последовательный порт (UART). При частоте 57600 Гц и настройке UART на 9 бит. Использовались подпрограммы и символические переменные в отдельном файле проекта.

4. Реализовать задержку программным способом, используя вызов подпрограммы.

Заданная задержка - 5.7с

Расчет погрешности:

$$\Delta x = 5.7 - 5.699861 = 0.000139;$$

$$x_{отн} = \frac{0.000139}{5.699871} 100\% = 0.0028\ \%.$$

Алгоритм основной программы:

1. Начало;
2. Вызываем программу задержки;
3. Конец.

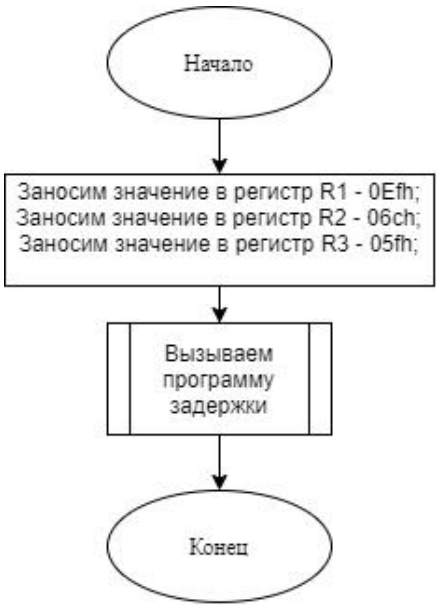
Алгоритм программы задержки

1. Начало;
2. Заносим значение в регистр R2 – 0E4h;
3. Заносим значение в регистр R3 – 0E2h;
4. Декремент регистра R3 и переход к п.2 не равен 0, иначе заносим значение 05ah в регистр R3 и переход к п.3;
5. Декремент регистра R2 и переход к п.2 не равен 0, иначе заносим значение 05bh в регистр R2 и переход к п.4;
6. Декремент регистра R1 и переход к п.2 не равен 0, иначе переход к п.5;
7. Заносим значение в регистр R2 – 0F1h;
8. Заносим значение в регистр R1 – 0F9h;
9. Декремент регистра R3 и переход к п.7 не равен 0, иначе заносим значение 0f1h в регистр R3 и переход к п.8;
10. Декремент регистра R2 и переход к п.7 не равен 0, иначе заносим значение 027h в регистр R2 и переход к п.9;

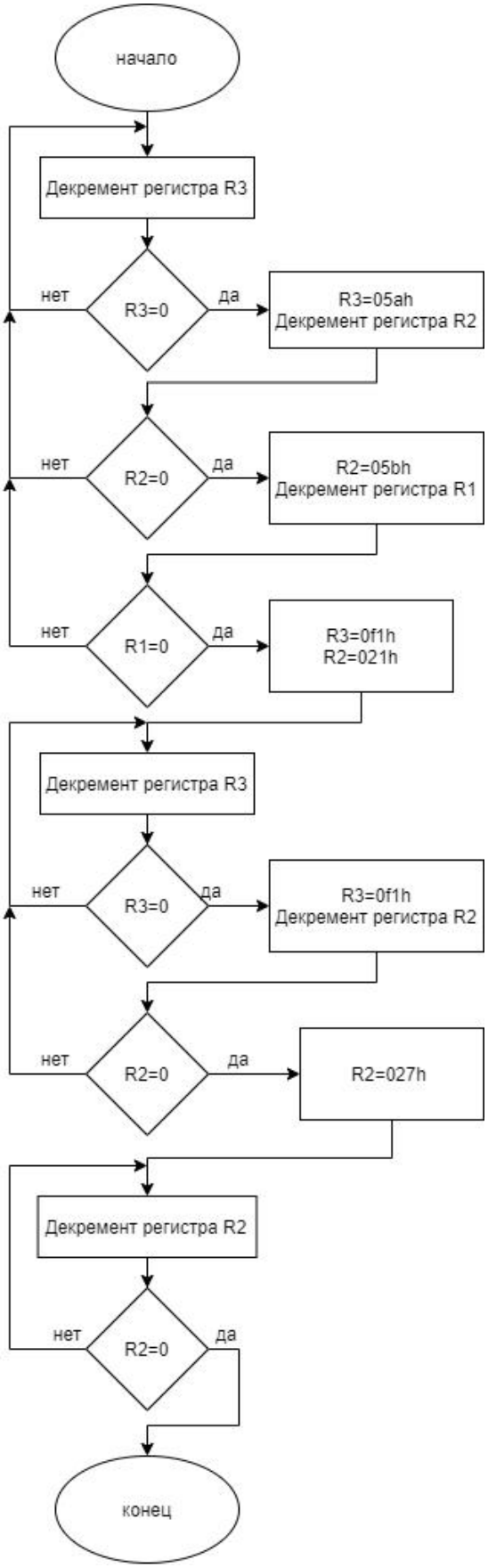
11. Декремент регистра R2 и переход к п.9 не равен 0, иначе переход к п.10;
12. Конец.

Блок-схема алгоритма:

Основная программа:



Подпрограмма DELAY:



Текст программы

Основная программа:

```
NAME MAIN_CODE
EXTRN CODE (DELAY)
MAIN SEGMENT CODE
CSEG AT 0
JMP start
PROG SEGMENT CODE
CONST SEGMENT CODE
STACK SEGMENT IDATA
RSEG STACK
        DS 10H; 16 BYTES STACK
        CSEG AT 0
        USING 0; REGISTER-BANK 0BITVAR SEGMENT BIT
RSEG PROG
start:
MOV R1, #02CH
MOV R2, #051H
MOV R3, #0FFH
CALL DELAY
JMP start
END
```

Подпрограмма DELAY:

```
NAME DELAY
PUBLIC DELAY
DELAY SEGMENT CODE
RSEG DELAY
del:
DJNZ R3, del
```

```
DJNZ R2, del
DJNZ R1, del
RET
END
```

Вывод: Задание выполнено успешно. В ходе выполнения работы были отлажены навыки по работе с подпрограммами и реализована задержка программным способом, используя вызов подпрограммы. частота работы МК равна 12 МГц. Точность длительности задержки выше, чем 0.01%. Точность длительности задержки = 0.0028%.

5. Реализовать задержку аппаратным способом, используя таймер T1 или T0 по выбору преподавателя. Обязательно использование прерываний.

Заданная задержка - 5.7с (T1)

Расчет погрешности:

$$\Delta x = 3.3 - 3.299999 = 0.000001;$$

$$x_{отн} = \frac{0.000001}{3.299999} 100\% = 0.00003\ \%.$$

Алгоритм основной программы:

1. Начало;
2. Вызываем программу задержки;
3. Конец.

Алгоритм программы задержки

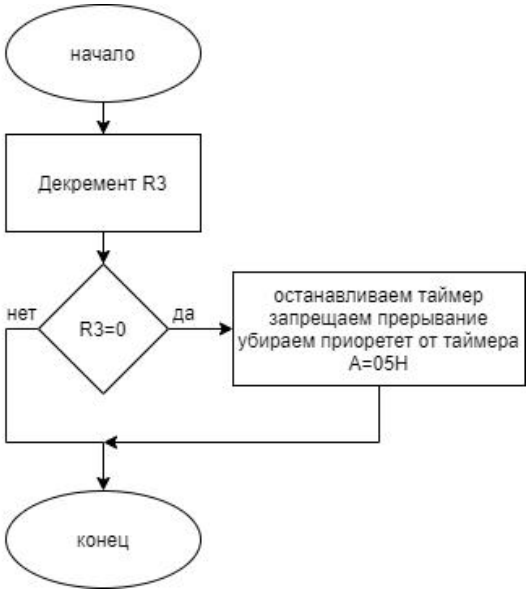
1. Начало;
2. Заносим значение в регистр R1 – 01Dh;
3. Заносим значение в регистр R2 – 0E4h;
4. Заносим значение в регистр R3 – 0E2h;
5. Декремент регистра R3 и переход к п.2 не равен 0, иначе заносим значение 05ah в регистр R3 и переход к п.3;
6. Декремент регистра R2 и переход к п.2 не равен 0, иначе заносим значение 05bh в регистр R2 и переход к п.4;
7. Декремент регистра R1 и переход к п.2 не равен 0, иначе переход к п.5;
8. Заносим значение в регистр R2 – 0F1h;

9. Заносим значение в регистр R1 – 0F9h;
10. Декремент регистра R3 и переход к п.7 не равен 0, иначе заносим значение 0f1h в регистр R3 и переход к п.8;
11. Декремент регистра R2 и переход к п.7 не равен 0, иначе заносим значение 027h в регистр R2 и переход к п.9;
12. Декремент регистра R2 и переход к п.9 не равен 0, иначе переход к п.10;
13. Конец.

Блок-схема алгоритма:

Основная программа:

Подпрограмма DELAY:



Текст программы

Основная программа:

```
NAME LAB1_5
EXTRN CODE (zaderjka)
MAIN SEGMENT CODE
CSEG AT 1
CALL start
CSEG AT 001bH
JMP obrabotchik
USING 1
RSEG MAIN
start:
CALL zaderjka
osnProg:
CJNE A , #00h, loop1
JMP osnProg
loop1:
JMP start
obrabotchik:
```

```
        DJNZ R3, resetTimer
        CLR TR1
        CLR IE.7
        CLR ET1
        CLR PT1
        mov a, #05h
        resetTimer:
RETI
END
```

Подпрограмма DELAY:

```
NAME zaderjka
PUBLIC        zaderjka
zaderjka SEGMENT CODE
RSEG zaderjka
DEC A
RET
END
```

Вывод: Задание выполнено успешно. В ходе выполнения лабораторной работы были получены навыки по работе с таймером/счетчиком и реализована задержка аппаратным способом, с использованием таймера. Частота работы МК равна 12 МГц. Точность длительности задержки выше, чем 0.01%. Точность длительности задержки = 0.00003 %.

6. Написать программу «Бегущие огни». Программа «Бегущие огни» должна выводить световые эффекты (не менее 16) с помощью 8 красных светодиодов. Эффекты переключаются двумя кнопками. При достижении максимального или минимального номера эффекта дважды по 0.5с должен засветиться зеленый светодиод. Другая пара кнопок должна регулировать скорость (не менее 16 скоростей) смены кадров в эффектах. При достижении минимальной или максимальной скорости должен засветиться зеленый светодиод на 1с. Если в течение 20с не нажата ни одна кнопка, программа автоматически должна переключать эффекты в сторону увеличения по циклу с выдержкой каждого эффекта не менее 30с.

Алгоритм основной программы:

1. Начало;
2. Установить начальные значения переменных N_EFFECTA, SPEED, DELAY, COUNT_CADR по 1;
3. Запуск таймера 0;
4. Инициализация таймеров и переменных;
5. Если нажата кн. 1, то переход на п.6, иначе переход на п.10;
6. Задержка от дребезга;
7. Обнуления 20 сек. интервала;
8. Декремент эффекта;
9. Если эффект минимальный, то засветить р.3.7 и запуск таймера 1 и переход п.5;
10. Если нажата кн. 2, то переход на п.11, иначе переход на п.15;
11. Задержка от дребезга;
12. Обнуления 20 сек. интервала;
13. Инкремент эффекта;
14. Если эффект максимальный, то засветить р.3.7 и запуск таймера 1 и переход п.5;
15. Если нажата кн. 3, то переход на п.16, иначе переход на п.20;
16. Задержка от дребезга;
17. Обнуления 20 сек. интервала;
18. Декремент скорости;
19. Если эффект минимальный, то засветить р.3.7 и запуск таймера 1 и

переход п.5;

20. Если нажата кн. 1, то переход на п.6, иначе переход на п.10;
21. Задержка от дребезга;
22. Обнуления 20 сек. интервала;
23. Декремент эффекта;
24. Если эффект минимальный, то засветить р.3.7 и запуск таймера 1 и переход п.5;
25. Если нажата кн. 4, то переход на п.26, иначе переход на п.5;
26. Задержка от дребезга;
27. Обнуления 20 сек. интервала;
28. Инкремент скорости;
29. Если эффект минимальный, то засветить р.3.7 и запуск таймера 1 и переход п.5;
30. Конец.

Алгоритм подпрограммы EFFECT(timer 0):

1. Начало;
2. Если кнопка не нажималась больше 20с, то переход на п.3, иначе переход на п.7;
3. Увеличить номер эффекта и установить счет при бездействии 30с, а не 20;
4. Если номер эффекта равен 16, то переход на п.4, иначе переход на п.7 ;
5. Установить время свечения зел. светодиода и засветить его
6. Запуск таймера 1;
7. Вызов SET_EFFECT;
8. декремент счетчик скорости;
9. Если счетчик скорости равен 0, то переход в п.10, иначе переход в п.11;
10. Вывод кадра;
11. Конец.

Алгоритм подпрограммы GREENOMER_EFECTA(timer 1):

12. Начало;

13. Сброс таймера 1;
14. Задание времени между вызовами прерываниями от таймера 1;
15. Если GR_TWO=2 то переход на п.16, иначе переход на п.18;
16. Задание задержки в 0.25 сек на паузу между свечениями;
17. Запуск таймера и переход на п.23;
18. Если GR_TWO=1 то переход на п.19, иначе переход на п.21;
19. Задание задержки в 0.5 сек на свечение ;
20. Запуск таймера и переход на п.23;
21. Если GR_TWO=1 то переход на п.22, иначе переход на п.23;
22. Остановка таймера 1;
23. Конец.

Алгоритм подпрограммы SET_EFFECT :

1. Начало;
2. Если NOMER_EFECTA=1, то переход на п.3, иначе переход на п.4
3. Занести в DPTR адрес 1 эффекта;
4. Если NOMER_EFECTA=2, то переход на п.5, иначе переход на п.6;
5. Занести в DPTR адрес 2 эффекта;
6. Если NOMER_EFECTA=3, то переход на п.7, иначе переход на п.8;
7. Занести в DPTR адрес 3 эффекта;
8. Если NOMER_EFECTA=4, то переход на п.9, иначе переход на п.10;
9. Занести в DPTR адрес 4 эффекта;
10. Если NOMER_EFECTA=5, то переход на п.11, иначе переход на п.12;
11. Занести в DPTR адрес 5 эффекта;
12. Если NOMER_EFECTA=6, то переход на п.13, иначе переход на п.14;
13. Занести в DPTR адрес 6 эффекта;
14. Если NOMER_EFECTA=7, то переход на п.15, иначе переход на п.16;
15. Занести в DPTR адрес 7 эффекта;
16. Если NOMER_EFECTA=8, то переход на п.17, иначе переход на п.18;
17. Занести в DPTR адрес 8 эффекта;
18. Если NOMER_EFECTA=9, то переход на п.19, иначе переход на п.20;
19. Занести в DPTR адрес 9 эффекта;
20. Если NOMER_EFECTA=10, то переход на п.21, иначе переход на п.22;

21. Занести в DPTR адрес 10 эффекта;
22. Если NOMER_EFECTA=11, то переход на п.23, иначе переход на п.24;
23. Занести в DPTR адрес 11 эффекта;
24. Если NOMER_EFECTA=12, то переход на п.25, иначе переход на п.26;
25. Занести в DPTR адрес 12 эффекта;
26. Если NOMER_EFECTA=13, то переход на п.27, иначе переход на п.28;
27. Занести в DPTR адрес 13 эффекта;
28. Если NOMER_EFECTA=14, то переход на п.29, иначе переход на п.30;
29. Занести в DPTR адрес 14 эффекта;
30. Если NOMER_EFECTA=15, то переход на п.31, иначе переход на п.32;
31. Занести в DPTR адрес 15 эффекта;
32. Если NOMER_EFECTA=16, то переход на п.33, иначе переход на п.34;
33. Занести в DPTR адрес 16 эффекта;
34. Конец.

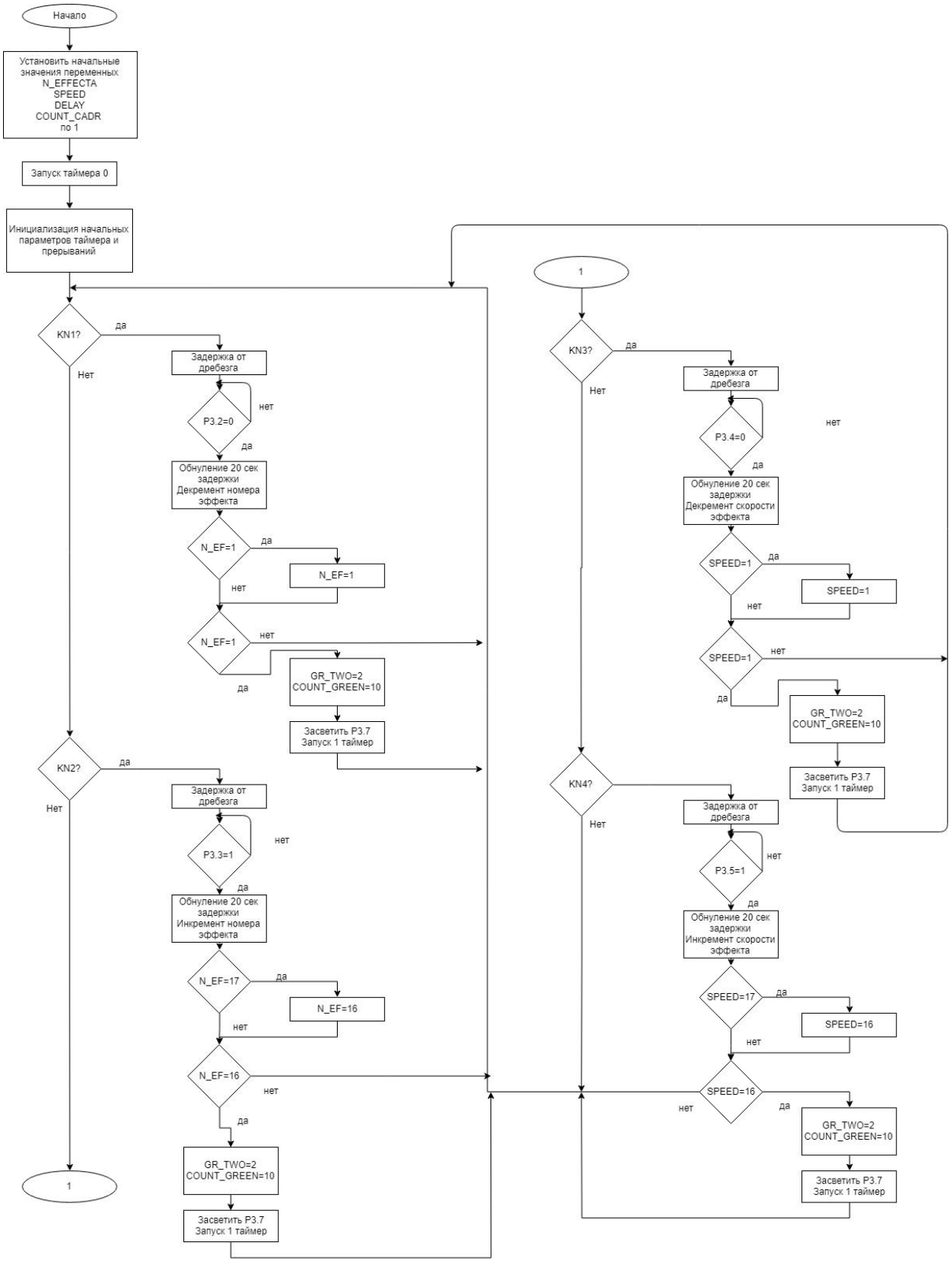
Алгоритм подпрограммы DELAY1:

1. Начало;
2. Занести в BIG 05;
3. Занести в R5 ffh;
4. Занести в R6 c4h;
5. Декремент R6;
6. Если R6=0 то переходим на п.7, иначе переходим на п.5;
7. Декремент R5;
8. Если R5=0 то переходим на п.9, иначе переходим на п.4;
9. Декремент BIG;
10. Если BIG=0 то переходим на п.10, иначе переходим на п.3;

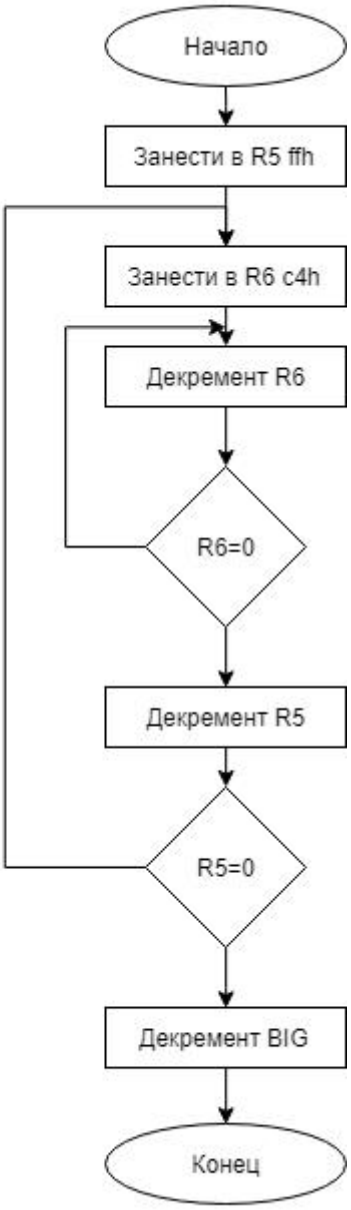
Конец.

Блок-схема алгоритма:

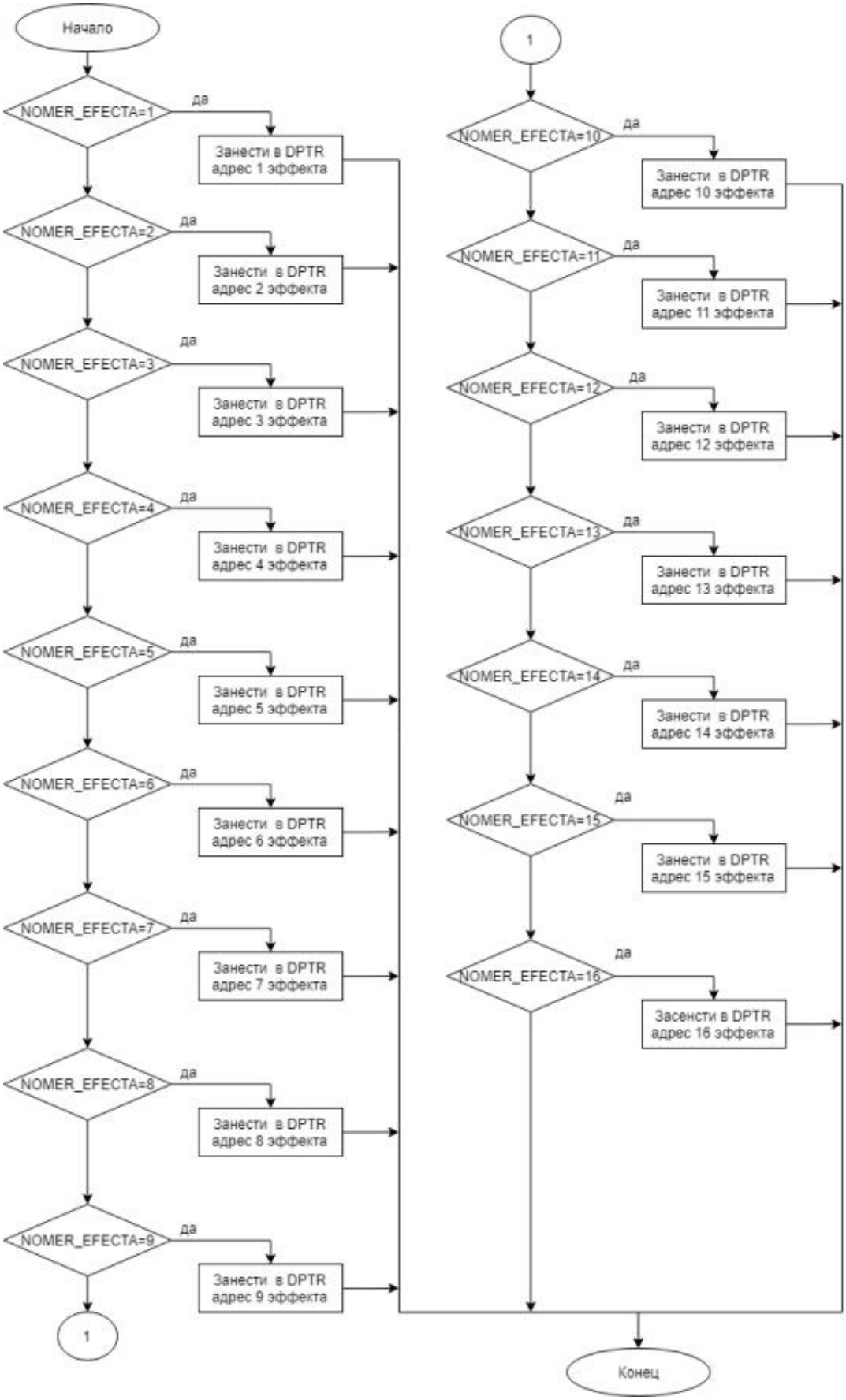
Основная программа:



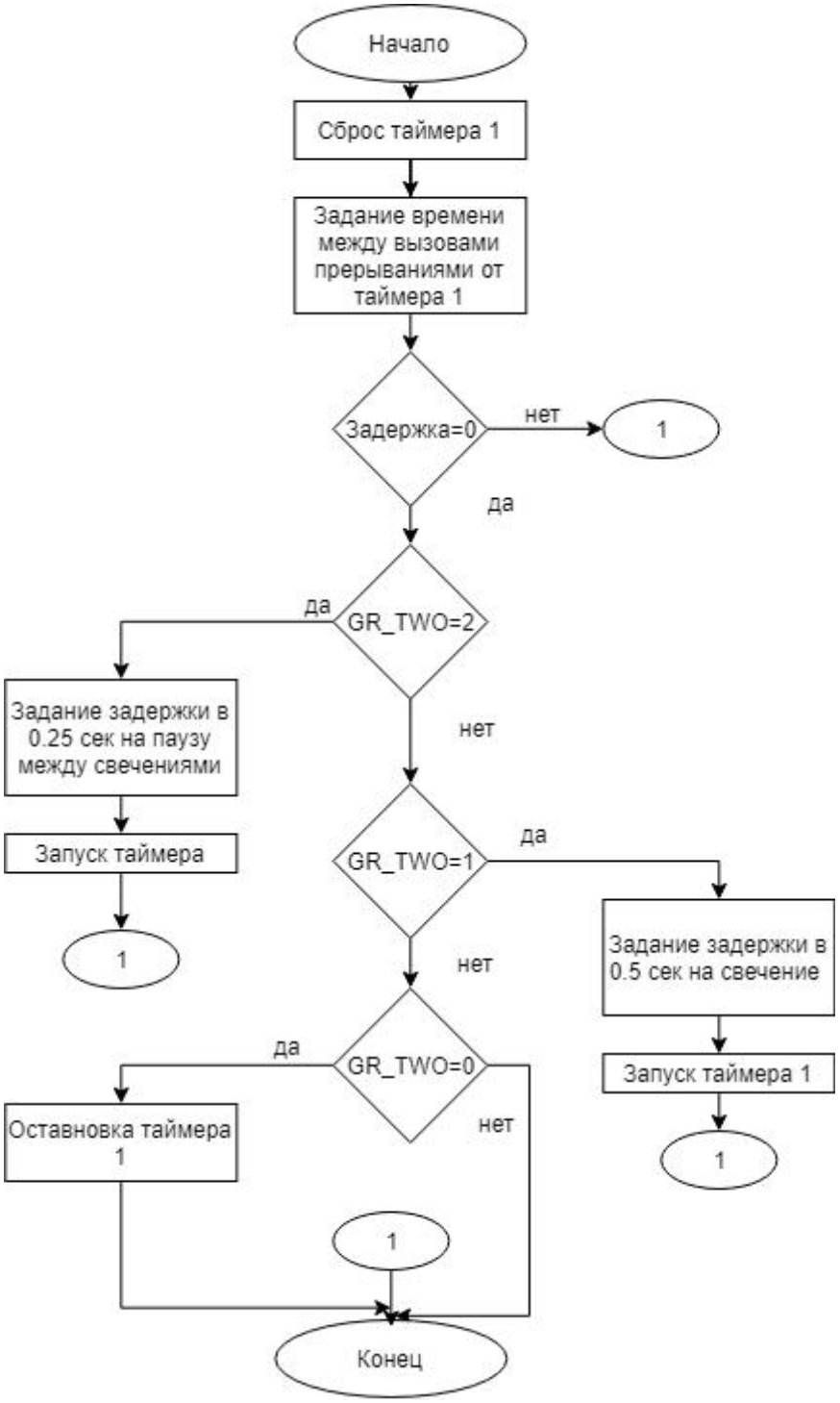
Подпрограмма DELAY1:



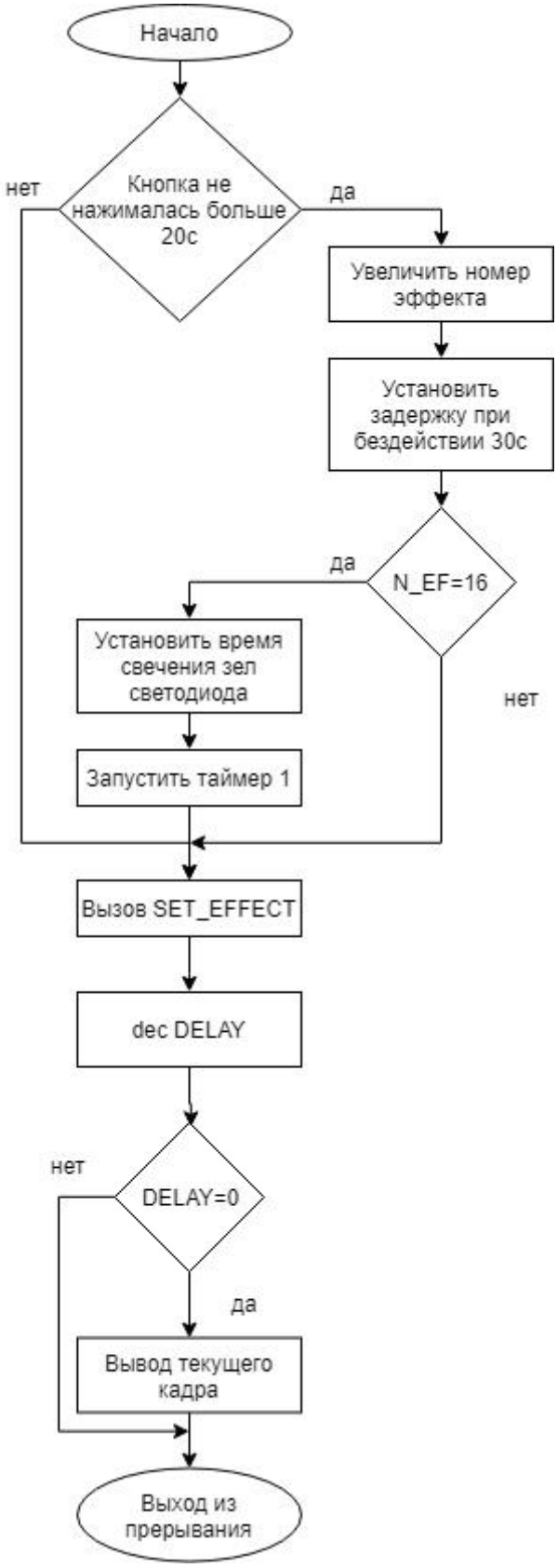
Подпрограмма SET_EFFECT:



Подпрограмма GREENOMER_EFFECTA:



Подпрограмма EFFECT:



Текст программы

```
NAME SUBB
USING 0                    ;банк регистров
BIG EQU 22H
ZADER20_H EQU 20H
ZADER20_L EQU 21H
GR_TWO      EQU    R7
COUNT_CADR EQU R0          ;счётчик кадров в эффекте
SPEED EQU R1               ;счётчик временных задержек
DELAY EQU R2               ;количество временных задержек
NOMER_EFECTA EQU R3        ;номер эффекта
COUNT_GREEN EQU R4         ;счётчик временных задержек
;эфекты
F1   SEGMENT CODE          ;декларировать перемещаемые сегменты различных типов в пространстве программ
F2   SEGMENT CODE
F3   SEGMENT CODE
F4   SEGMENT CODE
F5   SEGMENT CODE
F6   SEGMENT CODE
F7   SEGMENT CODE
F8   SEGMENT CODE
F9   SEGMENT CODE
F10 SEGMENT CODE
F11 SEGMENT CODE
F12 SEGMENT CODE
F13 SEGMENT CODE
F14 SEGMENT CODE
F15 SEGMENT CODE
```

```
F16 SEGMENT CODE
PROG SEGMENT CODE                ;объявление сегмента кода программы
STACK SEGMENT IDATA              ;объявление сегмента стека
RSEG STACK                       ;выбирает описанный перемещаемый сегмент и делает его активным
DS 16H                           ;под стек резервируется 16 байт
;DB резервирует необходимое количество байт в памяти программ
RSEG F1
        DB 55H, 0AAH, 55H, 0AAH, 55H, 0AAH, 55H, 0AAH
;DB резервирует необходимое количество байт в памяти программ
RSEG F2
        DB 3FH, 0CFH, 0F3H, 0FCH, 3FH, 0CFH, 0F3H, 0FCH
RSEG F3
        DB 0CCH, 033H, 0CCH, 033H, 0CCH, 033H, 0CCH, 033H
RSEG F4
        DB 0F0H, 0FH, 0F0H, 0FH, 0F0H, 0FH, 0F0H, 0FH
RSEG F5
        DB 7FH, 0BFH, 0DFH, 0EFH, 0F7H, 0FBH, 0FDH, 0FEH    ;мерганиеодного
RSEG F6
        DB 011H, 088H, 044H, 022H, 011H, 088H, 044H, 022H
RSEG F7
        DB 7EH, 3CH, 18H, 00H ,7EH, 3CH, 18H, 00H , 0FFH
RSEG F8
        DB 07EH, 03CH, 018H, 00H, 018H,03CH, 07EH, 00H
RSEG F9
        DB 00H, 018H, 03CH, 07EH, 0FFH, 07EH,03CH, 018H
RSEG F10
        DB 00H, 080H, 0C0H, 0E0H, 0F0H, 0F8H, 0FCH, 0FEH
RSEG F11
```

```
        DB 0FEH, 0FFH, 0FCH, 0FFH, 0F8H, 0FFH, 0F0H, 0FFH
RSEG F12
        DB 07FH, 0FFH, 03FH, 0FFH, 01FH, 0FFH, 0FH, 0FFH
RSEG F13
        DB    00H, 01H, 00H, 03H,00H, 07H, 00H, 0FH, 00H
RSEG F14
        DB 00H, 0FFH,00H, 0FFH, 00H, 0FFH, 00H, 0FFH,
RSEG F15
        DB 080H, 0C0H, 0E0H, 0F0H, 0F8H, 0FCH, 0FEH, 0FFH
RSEG F16
        DB 018H, 081H, 03CH, 0C3H, 018H, 081H, 03CH, 0C3H
CSEG AT 0
    LJMP go                         ;при включении питания будет произведён безусловный переход на метку START
;ORG - используется для указания ассемблеру адреса объекта в памяти
ORG 0BH                             ;адрес вектора прерываний от T\C0
CALL EFFECT                         ;вызвать подпрограмму, выводящую кадр эффекта
RETI
ORG 1BH                             ;адрес вектора прерываний от T\C1
CALL GREENOMER_EFECTA               ;вызвать подпрограмму, обрабатывающую свечение зелёного светодиода
RETI
RSEG PROG                           ;сегмент кода программы:
DELAY1:                             ;ЗАДЕРЖКА
        MOV BIG, #05
        MOV R5, #0FFH
        MMMM1:
                MOV R6, #0FFH
                MMMM2:
                DJNZ R6, MMMM2
```

```
        DJNZ R5, MMMM1
        MOV R5, #0FFH
        DJNZ BIG, MMMM1
RET
GREENOMER_EFECTA:                ;задержка на свечение зеленого светодиода
        CLR TCON.6
MOV TH1, #03CH
MOV TL1, #0AFH
DJNZ COUNT_GREEN, ST_GREEN
            CJNE GR_TWO, #02H, LIGHT
                MOV COUNT_GREEN, #5
                DEC GR_TWO
                SETB P3.7
            LJMP ST_GREEN
            LIGHT:
            CJNE GR_TWO, #01H, OUT
                MOV COUNT_GREEN, #10
                DEC GR_TWO
                CLR P3.7
            LJMP ST_GREEN
            OUT:
            CJNE GR_TWO, #00H, END_GREEN
               SETB P3.7
               LJMP END_GREEN
    ST_GREEN:
          SETB TCON.6
 END_GREEN:
RET
SET_EFFECT:                      ;ВЫБОР ЭФФЕКТА
```

```
        CJNE NOMER_EFECTA, #1, L1   ;Сравнение аккумулятора с прямоадресуемым байтом и переход, если не равно
                MOV DPTR, #F1 -  1          ;DPTR адресные регистры
L1:   CJNE NOMER_EFECTA, #2, L2
                MOV DPTR, #F2 -  1
L2:   CJNE NOMER_EFECTA, #3, L3
                MOV DPTR, #F3 -  1
L3:   CJNE NOMER_EFECTA, #4, L4
                MOV DPTR, #F4 -  1
L4:   CJNE NOMER_EFECTA, #5,L5
                MOV DPTR, #F5 -  1
L5:   CJNE NOMER_EFECTA, #6, L6
                MOV DPTR, #F6 -  1
L6:   CJNE NOMER_EFECTA, #7, L7
                MOV DPTR, #F7 -  1
L7:   CJNE NOMER_EFECTA, #8, L8
                MOV DPTR, #F8 -  1
L8:   CJNE NOMER_EFECTA, #9, L9
                MOV DPTR, #F9 -  1
L9:   CJNE NOMER_EFECTA, #10,L10
                MOV DPTR, #F10 - 1
L10: CJNE NOMER_EFECTA, #11,L11
                MOV DPTR, #F11 - 1
L11: CJNE NOMER_EFECTA, #12,L12
                MOV DPTR, #F12 - 1
L12: CJNE NOMER_EFECTA, #13,L13
                MOV DPTR, #F13 - 1
L13: CJNE NOMER_EFECTA, #14,L14
                MOV DPTR, #F14 - 1
```

```
L14: CJNE NOMER_EFECTA, #15, L15
        MOV DPTR, #F15 - 1
L15: CJNE NOMER_EFECTA, #16, L16
        MOV DPTR, #F16 - 1
L16: RET
EFFECT:                                          ;вывод кадра:
    DJNZ ZADER20_L,SKIP
        MOV ZADER20_L, #255
        DJNZ ZADER20_H,SKIP
        INC NOMER_EFECTA
        MOV ZADER20_L,#190
        MOV ZADER20_H,#2
        CJNE NOMER_EFECTA, #17, M11      ;если номер эффекта не равен 17 то переход
        MOV NOMER_EFECTA, #16
        M11:
            CJNE NOMER_EFECTA, #16, SKIP
                MOV COUNT_GREEN, #10
                MOV GR_TWO, #02H
                CLR P3.7
    ;зелёный светодиод начинает светиться
                SETB TCON.6
    ;запускается первый таймер
    SKIP:
CALL SET_EFFECT                  ;вызов эфекта
DJNZ DELAY, END_CADR             ;уменьшаем счётчик задержек (Декремент регистра и переход, если не нуль)
MOV A, SPEED                     ;если счётчик задержек достиг нуля, присваиваем ему начальное значение
MOV DELAY, A
MOV A, DPL
```

```
ADD A, COUNT_CADR          ;прибавляем к базовому адресу эффекта счётчик кадров, получаем адрес текущего кадра
MOV DPL, A
MOV A, DPH
ADDC A,#0
MOV DPH, A
CLR A                      ;Сброс аккумулятора
MOVC    A, @A+DPTR
MOV P1, A                  ;выводим кадр
INC COUNT_CADR
CJNE COUNT_CADR, #9, END_CADR
MOV COUNT_CADR, #1
END_CADR:
RET
go:                                ; основная программа:
MOV GR_TWO, #02H
MOV SP, #STACK-1           ; инициализация стека!
MOV ZADER20_L, #40
MOV ZADER20_H, #2
MOV TMOD, #11H             ;T/C0 и T/C1 работают в режиме 16-битных таймеров
SETB IE.7                          ; разрешаются прерывания
SETB IE.1                          ;разрешается прерывание от нулевого таймера
SETB IE.3                          ; разрешается прерывание от первого таймера
MOV TH0, #0H
MOV TL0, #0H
SETB TCON.4
CLR    TCON.6
MOV TH1, #03CH
```

```
MOV TL1, #0AFH
MOV NOMER_EFECTA, #1        ;устанавливается номер эффекта - первый
MOV SPEED, #1               ;устанавливается количество временных задержек
MOV DELAY, #1               ;устанавливаем счётчик задержек
MOV COUNT_CADR,  #1         ;устанавливаем счётчик кадров
KN1:
    JB P3.2, KN2            ;если не нажата первая кнопка, перйти к опросу второй
    CALL DELAY1
    JNB P3.2, $             ;если не подтянут к 0 то переход на СЕБЯ
    MOV ZADER20_L, #40
  MOV ZADER20_H, #2

    DEC  NOMER_EFECTA
    CJNE NOMER_EFECTA, #0, M    ;если номер эффекта не равен нулю то переход
    MOV NOMER_EFECTA, #1
    M:
        CJNE NOMER_EFECTA, #1, KN1      ;если номер эффекта не равен 1 то переход KN1 и дальше опрашиваем все кнопки
        MOV GR_TWO, #02H                ;для двойного мерцания
        MOV COUNT_GREEN, #10
        CLR P3.7                ;зелёный светодиод начинает светиться
        SETB TCON.6
    ;запускается первый таймер

KN2:
    JB P3.3, KN3                ;если не нажата вторая кнопка, перйти к опросу третьей
    CALL DELAY1
```

```
    JNB P3.3, $                    ;если не подтянут к 0 то переход на последнюю метку


   MOV ZADER20_L, #40
   MOV ZADER20_H, #2


      INC   NOMER_EFECTA
    CJNE NOMER_EFECTA, #17, M1     ;если номер эффекта не равен 17 то переход
     MOV NOMER_EFECTA, #16
      M1:
          CJNE NOMER_EFECTA, #16, KN2   ;если номер эффекта не равен 6 то переход KN2 и дальше опрашиваем все кнопки
          MOV GR_TWO, #02H              ;для двойного мерцания
          MOV COUNT_GREEN, #10
          CLR P3.7                      ;зелёный светодиод начинает светиться
          SETB TCON.6                   ;запускается первый таймер
KN3:
       JB P3.4, KN4
       CALL DELAY1
JNB P3.4, $


MOV ZADER20_L, #40
MOV ZADER20_H, #2
       DEC SPEED
CJNE SPEED, #0, LL
MOV SPEED, #1
LL:   CJNE SPEED, #1, KN3
          MOV GR_TWO, #0             ;для отключения двойного мерцания
```

```
        MOV COUNT_GREEN, #20
        CLR P3.7                    ;зелёный светодиод начинает светиться
        SETB TCON.6                 ;запускается первый таймер
KN4:
        JB P3.5, KN1                ;если не нажата четвёртая кнопка, перейти к опросу первой
        CALL DELAY1
        JNB P3.5, $
        MOV ZADER20_L, #40
        MOV ZADER20_H, #2
        INC SPEED
        CJNE SPEED, #17, LL1
        MOV SPEED, #16
LL1:    CJNE SPEED, #16, KN4
        MOV GR_TWO, #0;для отключения двойного мерцания
        MOV COUNT_GREEN, #20
        CLR P3.7; зелёный светодиод начинает светиться
        SETB TCON.6; запускается первый таймер
        JMP KN1
JMP go
END
```

Вывод: Задание выполнено успешно. В ходе работы были комплексно использованы навыки, полученные в предыдущих пунктах и реализована программа «Бегущие огни». Был реализован весь функционал программы.

Вывод: в ходе выполнения лабораторной работы была изучена архитектура и организация микроконтроллеров семейства Intel 8051. Получены навыки программирования, тестирования и отладки устройств на базе микроконтроллеров семейства MCS-51 с использованием среды разработки ProView32 и программатора ProAtMic.