Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт по лабораторной работе №1

По дисциплине: «Аппаратные средства вычислительной техники»

Тема: «Битовый процессор микроконтроллера Intel 8051»

Вариант №3

Выполнил: Березин М.А.
студент группы ИУ8-73

Проверил: Рафиков А. Г.,
Старший преподаватель
кафедры ИУ

г. Москва,

2021 г.

1. Цель работы

В ходе лабораторной работы студент должен изучить основы работы с ассемблером ASM51, микроконтроллером Intel 8051 и его битовым процессором.

2. Ход работы

В рамках лабораторной работы необходимо реализовать три программы на ассемблере ASM51.

Булева функция восьми переменных, которую надлежит реализовать в рамках задания:

$$Q = \left(W \cdot C + X \cdot \overline{Z}\right) + (A + C) \cdot (U + B)$$

Расположение входных и выходных данных в памяти микроконтроллера показано в таблице 1:

Q	W	V	X	Z	A	C	U	B
P3.2	22H.0	P1.3	28H.2	P2.5	28H.5	25H.0	20H.3	21H.4

Таблица 1 – Расположение данных в памяти

Вычислить заданную функцию требуется тремя методами:

• С использованием только безусловных и условных переходов (с битовыми условиями)

• С использованием только битовых операций

• Без использования битовых операций и условных переходов с битовыми условиями (т.е. с использованием байтовых операций и переходов с байтовыми условиями)

Проверка работы программ производится путём моделирования в среде моделирования Proteus.

Для простоты кодирования алгоритма представим его в виде схемы из функциональных элементов, как показано на рисунке 1.

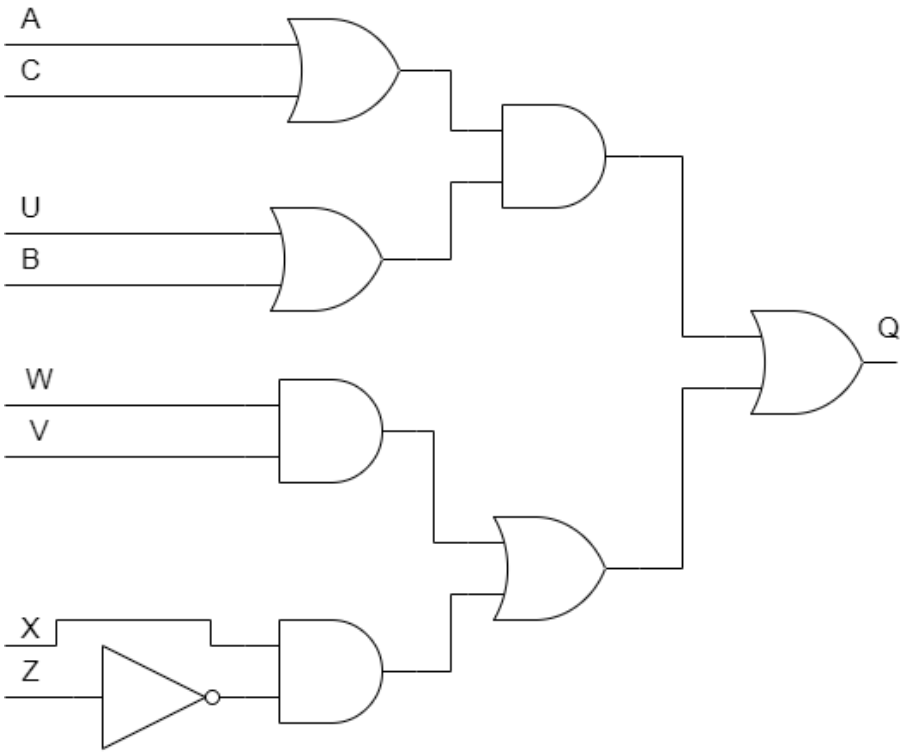


Рисунок 1 – Заданная по условию булева функция

Результат вычисления функции отображается визуально при помощи синего светодиода, параметры цепи: $U = 5\text{В}, I = 10^{-2}A, U_{\text{пад}} = 2.2\text{В}, U_T = 0.2\text{В}$

Рассчитаем требуемый для него резистор.

$$R = \frac{U - U_{\text{пад}} - U_T}{I} = \frac{(5 - 2.2 - 0.2)\text{В}}{10^{-2}\text{А}} = 260\ \text{Ом}$$

3. Практическая часть

В рамках лабораторной работы в среде моделирования Proteus была реализована схема для проверки кода на ассемблере ASM51. Схема приведена в приложении А. Состав схемы:

• Устройство ввода-вывода, необходимое для выбора значений булевых переменных (интерактивные константы)

• Устройство отображения результата работы функции (светодиод)

• Микроконтроллер I8051

Схема построена так, что, что логический нуль соответствуют горящему светодиоду.

Для всех трёх программ реализуем общую начальную часть программы, которая определяет короткие имена для использованных битов, а также исполняет инструкцию org, отвечающую за определение размещения программы в памяти – листинг кода показан на рисунке 2.

```
4
5   Q BIT P3.2
6   A1 BIT 28h.5 ; P0.2
7   B1 BIT 21h.4 ; P0.5
8   C1 BIT 25h.0; P0.3
9   U BIT 20h.3; P0.4
10  V BIT P1.3
11  W BIT 22h.0; P0.0
12  X BIT 28h.2;  P0.1
13  Z BIT P2.5
14
15  F1 BIT 20h.2
16
17  A_ BIT P0.2
18  B_ BIT P0.5
19  C_ BIT P0.3
20  U_ BIT P0.4
21  W_ BIT P0.0
22  X_ BIT P0.1
23
```

Рисунок 2 – Инициализация имён

С использованием битовых логических операций ANL, CPL, ORL и операции MOV реализуем программу, которая решает поставленную задачу с ограничением до категории команд битового процессора (кроме переходов). Листинг её кода (кроме общей части) приведён на рисунке 3.

```
org 100h

AGAIN:      ; AGAIN SUBROUTINE
    MOV C, A_
    MOV A1, C
    MOV C, B_
    MOV B1, C
    MOV C, C_
    MOV C1, C
    MOV C, U_
    MOV U, C
    MOV C, W_
    MOV W, C
    MOV C, X_
    MOV X, C
    SETB P2.0
    MOV C,A1
    ORL C,C1
    MOV F0,C
    MOV C,U
    ORL C,B1
    ANL C,F0
    MOV F0,C
    MOV C,W
    ANL C,V
    MOV F1,C
    MOV C,X
    ANL C,/Z
    ORL C,F1
    ORL C,F0
    CPL C
    MOV Q,C
    SJMP AGAIN ; DO CONTINEOUSLY
END
```

Рисунок 3 – Реализация на битовых командах

Реализуем эту же программу с использованием безусловных и условных (по битовом условию) переходов и только их. Для этого построим вспомогательную блок-схему (приведена в приложении В), после чего закодируем полученный алгоритм. Листинг реализации приведён на рисунке 4.

```
org 100h

AGAIN:      ; AGAIN SUBROUTINE
    MOV C, A_
    MOV A1, C
    MOV C, B_
    MOV B1, C
    MOV C, C_
    MOV C1, C
    MOV C, U_
    MOV U, C
    MOV C, W_
    MOV W, C
    MOV C, X_
    MOV X, C


TEST_W:  JNB    W, TEST_X
TEST_V:   JB V, CLR_Q

TEST_X:  JNB X, TEST_A
TEST_Z:  JNB Z, CLR_Q

TEST_A:  JB A1, TEST_U
TEST_C:  JNB C1, SET_Q

TEST_U:  JB U, CLR_Q
TEST_B:      JNB B1, SET_Q

CLR_Q:   CLR Q
         JMP AGAIN

SET_Q:   SETB Q
          JMP AGAIN

END
```

Рисунок 4 – Реализация на условных и безусловных переходах

На основе реализации с переходами с битовыми условиями сделаем аналогичную реализацию с переходами по байтовому условию. Листинг реализации представлен в приложении Г.

4. Тестирование реализованного алгоритма при помощи схемы тестирования в программе моделирования Proteus

Для тестирования реализованных алгоритмов, в первой и второй задаче данные подаются извне на все перечисленные переменные.

В третьем задании подаются только лишь 2 входа, которым не соответствуют внутренние ячейки памяти. Для тех же, кому соответствуют внутренние ячейки памяти, инициализация происходит программным способом напрямую инициализируя те или иные байты памяти. Листинг программы приведен в приложении Б.

В третьем задании переменные, которые не подаются через порты, инициализируются следующим образом:

A = 1

X = 1

B = 0

C = 1

U = 0

W = 1

Посредством данной инициализации мы имеем возможность изменять 2 входных значения V и Z. И получать 4 различных вариантов входных данных.

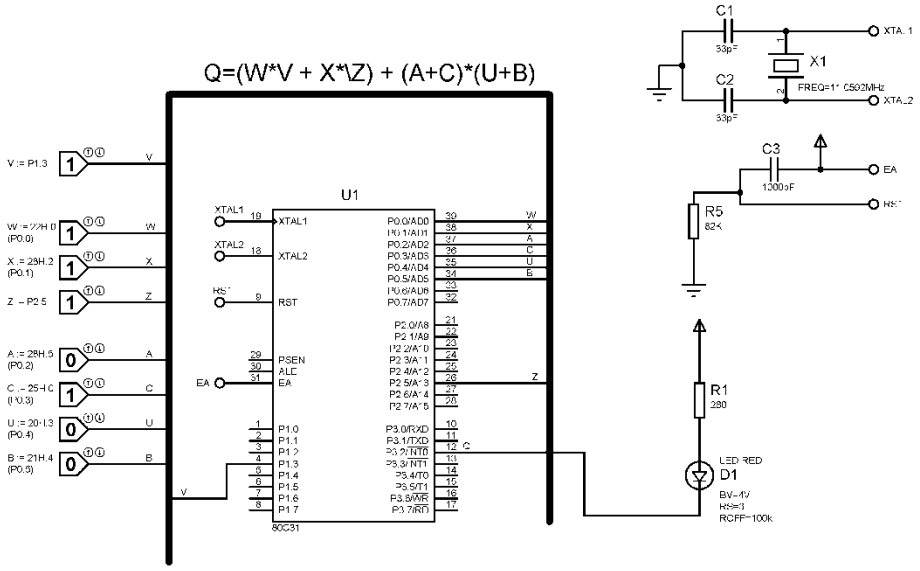
5. Выводы

В ходе лабораторной работы были изучены основы работы с ассемблером ASM51, микроконтроллером Intel 8051 и его битовым процессором.

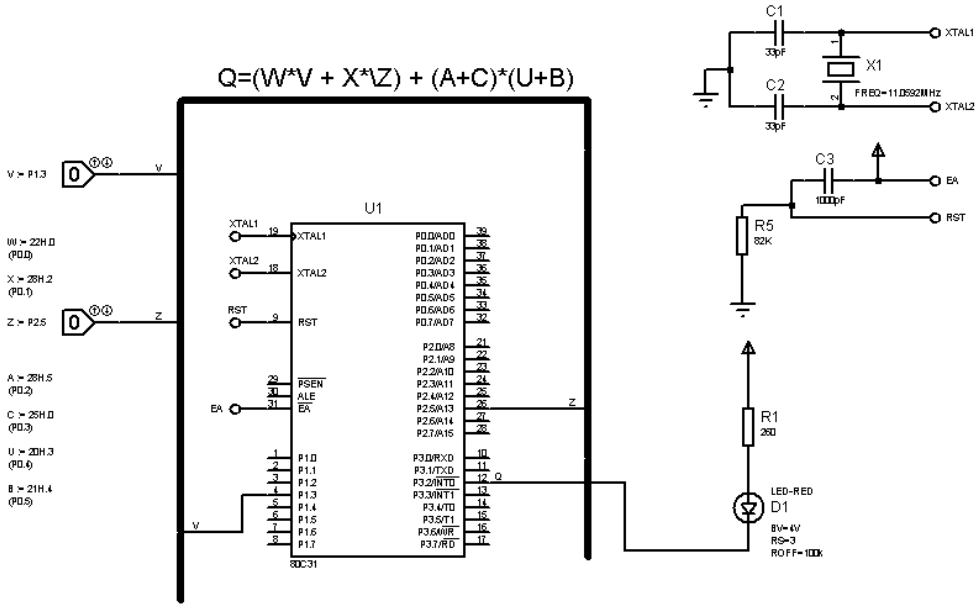
Результаты тестирования трёх реализованных алгоритмов совпадают с аналитическим решением, что означает, что задание выполнено корректно.

На основании анализа реализованных алгоритмов на ассемблере ASM51 установлено, что для решения битовых задач битовый процессор эффективнее и проще в применении, чем побайтовые операции.

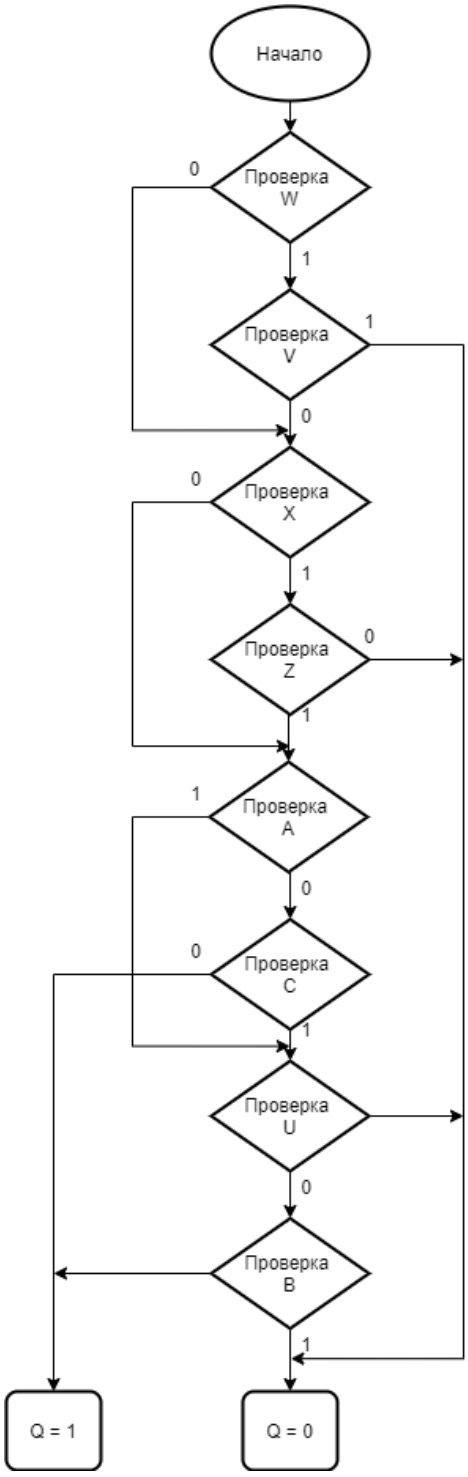
Приложение А. Схема тестирования программы на ASM51 (1 и 2)



Приложение Б. Схема тестирования программы на ASM51 (3 задача)



Приложение В. Блок-схема для реализации переходов



Приложение Г. Листинг для задания 3

```
; $NOMOD51
; $INCLUDE (8051.MCU)

      ;A1 BIT 28h.5 ; P0.2
       ;X BIT 28h.2;  P0.1
       ;B1 BIT 21h.4 ; P0.5
       ;C1 BIT 25h.0; P0.3
       ;U BIT 20h.3; P0.4
       ;W BIT 22h.0; P0.0

       ;V BIT P1.3
       ;Z BIT P2.5
       Q BIT P3.2

      ;A_ BIT P0.2
       ;B_ BIT P0.5
       ;C_ BIT P0.3
       ;U_ BIT P0.4
       ;W_ BIT P0.0
       ;X_ BIT P0.1

       org 100h
       AGAIN:            ; AGAIN SUBROUTINE

              ;MOV ACC, P0
              ;ANL ACC, #00000110b
              ;MOV 28h, ACC
              ;
              ;MOV ACC, P0
              ;ANL ACC, #00100000b
              ;MOV 21h, ACC
              ;
              ;MOV ACC, P0
              ;ANL ACC, #00001000b
              ;MOV 25h, ACC
              ;
              ;MOV ACC, P0
              ;ANL ACC, #00100000b
```

```
                ;MOV 20h, ACC

                MOV 28H, #00100100B ; 5b - A; 2b - X  #00100100B
                MOV 21H, #00000000B ; 4b - B                #00010000B
                MOV 25H, #00000001B ; 0b - C                #00000001B
                MOV 20H, #00000000B ; 3b - U                #00001000B
                MOV 22H, #00000001B ; 0b - W                #00000001B

TEST_W:         MOV ACC, 22H
                ANL ACC, #00000001B
                JZ TEST_X

TEST_V:          MOV ACC, P1
                 ANL ACC, #00001000B
                 JNZ  CLR_Q

TEST_X:          MOV ACC, 28H
                 ANL ACC, #00000100B
                 JZ TEST_A

TEST_Z:          MOV ACC, P2
                 ANL ACC, #00100000B
                 JZ CLR_Q

TEST_A:          MOV ACC, 28H
                 ANL ACC, #00100000B
                 JNZ TEST_U

TEST_C:          MOV ACC, 25h
                 ANL ACC, #00000001B
                 JZ SET_Q

TEST_U:          MOV ACC, 20H
                 ANL ACC, #00001000B
                 JNZ CLR_Q

TEST_B:          MOV ACC, 21H
                 ANL ACC, #00100000B
                 JZ SET_Q

;CLR_Q:      CLR Q
```

```
;             JMP AGAIN
;
;SET_Q:     SETB Q
;             JMP AGAIN

CLR_Q:     MOV P3, #00000000B
             JMP AGAIN

SET_Q:     MOV P3, #00000100B
             JMP AGAIN
END
```