ГУАП

КАФЕДРА № 44

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

канд. техн. наук		Т.Н.Соловьёва
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

АРХИТЕКТУРА ЯДРА И СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРОВ MCS-51

по курсу: МИКРОКОНТРОЛЛЕРНЫЕ СИСТЕМЫ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4143		Е.Д.Тегай
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

Цель работы

Изучение архитектуры ядра и системы команд микроконтроллеров семейства MCS-51; приобретение навыков программирования микроконтроллеров.

Задание по работе

Необходимо разработать три программы на языке ассемблера MCS-51:

1) программу для вычисления заданного арифметического выражения (для всех операциях полагайте, что операнды и результат – целые однобайтные числа; результат вычислений разместите в ячейке внутренней памяти данных 30h);

2) программу для записи заданного массива чисел во внешнюю память данных;

3) программу на ассемблере битового процессора для вычисления заданного логического выражения (результат выполнения разместите в любой ячейке памяти данных битового процессора).

Работу программ необходимо проверить с помощью симулятора.

Задание индивидуального варианта продемонстрировано на рисунке 1, где в первой ячейке показан номер индивидуального варианта, а в ячейках 2 – 4 показаны соответственно задания для задач 1 – 3.

9	$\frac{1}{2}(X-Z)\left(Y+\frac{X}{Z}\right)$	0h…15h…0h…15h…0h	$(\bar{x} \oplus yr) \vee d$

Рисунок 1 – Индивидуальное задание

Разработка программы 1

Задумка алгоритма работы программы довольно проста. Для начала необходимо обозначить и идентифицировать некоторыми числовыми значениями переменные X, Y и _Z. Обозначение проводилось с помощью команды ***equ***, где в результате для каждой переменной (не только для перечисленных трёх, но и для промежуточных и результирующих, о которых далее) было объявлено своё место в памяти. Выбор места определялся

«случайным» в некотором смысле образом, не влияющим на суть логики программы.

Так как выражение состоит из нескольких математических операций, следует завести пару промежуточных переменных (buf1,buf2), которые будут хранить в себе значение какой-либо операции для дальнейшего использования. Было выбрано именно две промежуточных переменных только потому, что с помощью одной довольно неудобно создавать программу. Также для хранения результата была добавлена переменная rez.

Затем нужно задать начальные числовые значения для исходных переменных X, Y и _Z. Выбор значения осуществлялся случайным образом.

На рисунке 2 изображён алгоритм работы программы с помощью порядковой нумерации математических операций и логики работы промежуточных переменных.

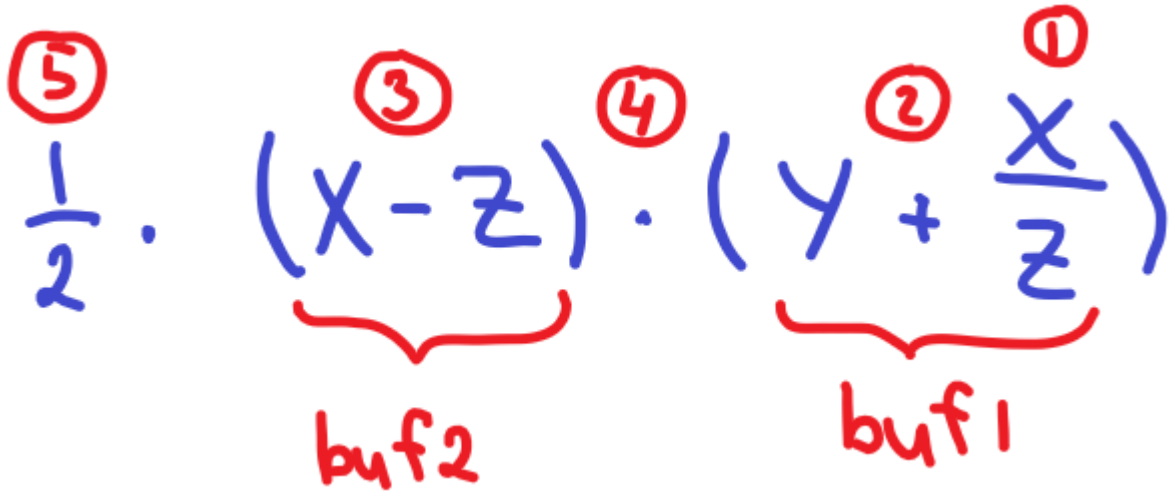

Рисунок 2 – Алгоритм работы программы

Математические операции проводились с помощью команд: ***div, add, sub*** и ***mul***. Также следует отметить, что для корректной работы некоторых команд использовались аккумуляторы A и B, а также для наглядности работы использовались регистры R0, R1 и R2.

Текст программы

```
;*****************************
;Filename: lb_1_S6_Tegai_4143
;Date: 2024/02/05
```

```
;File Version: 1
;Author: Tegai E.D.
;Company: SUAI
;Description: lb_1
;*******************************
;Переменные
;*******************************
X equ 0h  ; адрес для X
Y equ 01h  ; адрес для Y
_Z equ 02h  ; адрес для _Z
buf1 equ 08h ; адрес для промежуточного значения buf1
buf2 equ 09h ;  адрес для промежуточного значения buf2
rez equ 0Ah ; адрес для результата rez
;*******************************
; Reset vector
org     0h      ; process reset vector
ajmp  start   ; go to beginning of program
;*******************************
; Код
;*******************************
MAIN:
   org          100h  ; starting address in external data memory
;*******************************
; 1/2 * (X - Z) * (Y + X/Z)
;*******************************
start:
   mov X, #0Ch ; передаём числовое значение в переменную X
   mov Y, #06h ; передаём числовое значение в переменную Y
   mov _Z, #02h ; передаём числовое значение в переменную _Z
```

```
;(Y + X/Z)
mov A, X  ; передаём на аккумулятор A значение X
mov B, _Z ; передаём на аккумулятор B значение _Z
div AB  ; делим значение на аккумуляторе A на значение на аккумуляторе B (результат деления на аккумуляторе A)
add A, Y  ; прибавляем к полученному частному Y
mov buf1, A ; передаём значение суммы во временную переменную

;(X - Z)
mov A, X ; передаём на аккумулятор A значение X
subb A, _Z ; вычиитаем из значения на аккумуляторе A значение переменной _Z
mov buf2, A ; передаём полученную разность во временную переменную

; получаем итоговое выражение
mov A, buf1 ; передаём на аккумулятор A значение временной переменной
mov B, buf2 ; передаём на аккумулятор A значение временной переменной
mul AB ; умножаем значения на аккумуляторах (результат на аккумуляторе A)
mov B, _Z ; передаём значение переменной _Z на аккумулятор B
div AB ; делим значение на аккумуляторе A на значение на аккумуляторе B

mov rez, A ; передаём полученный результат в соответствующую результирующую переменную

sjmp $ ; бесконечный цикл

END
```

Результат работы программы

Результаты работы программы продемонстрированы на рисунках 3 - 9.

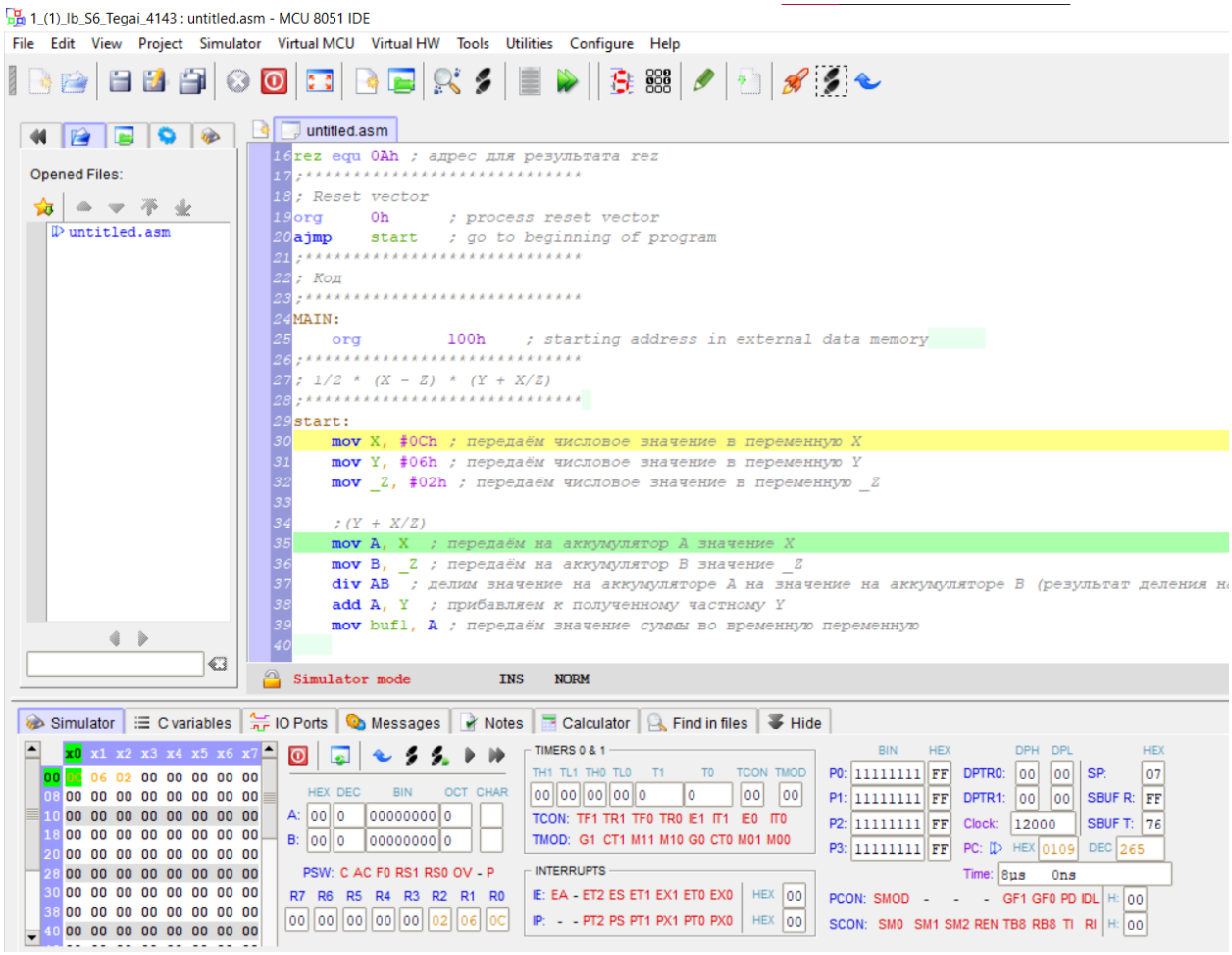

Рисунок 3 – Вывод числовых значений X,Y,_Z

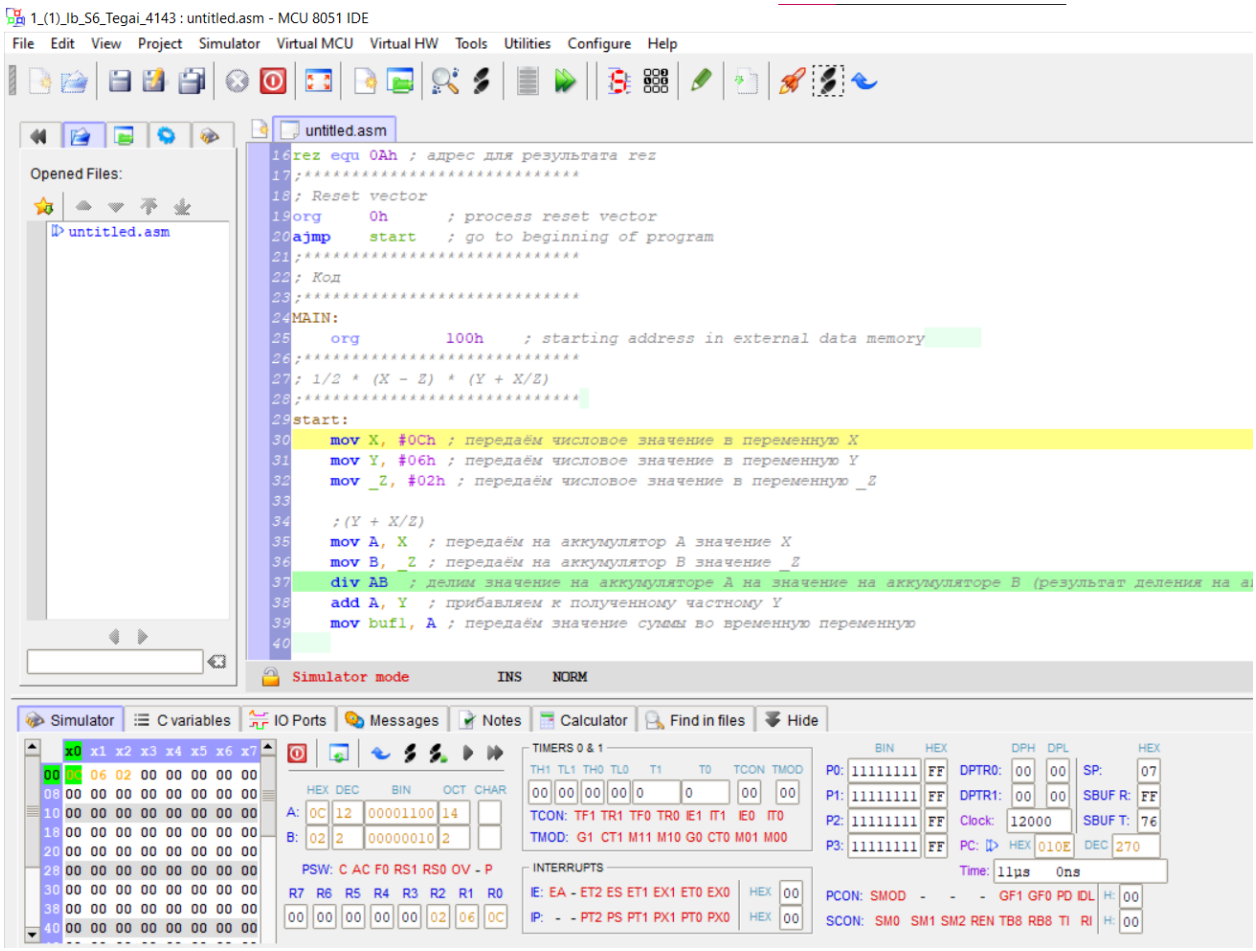

Рисунок 4 – Заполнение аккумуляторов

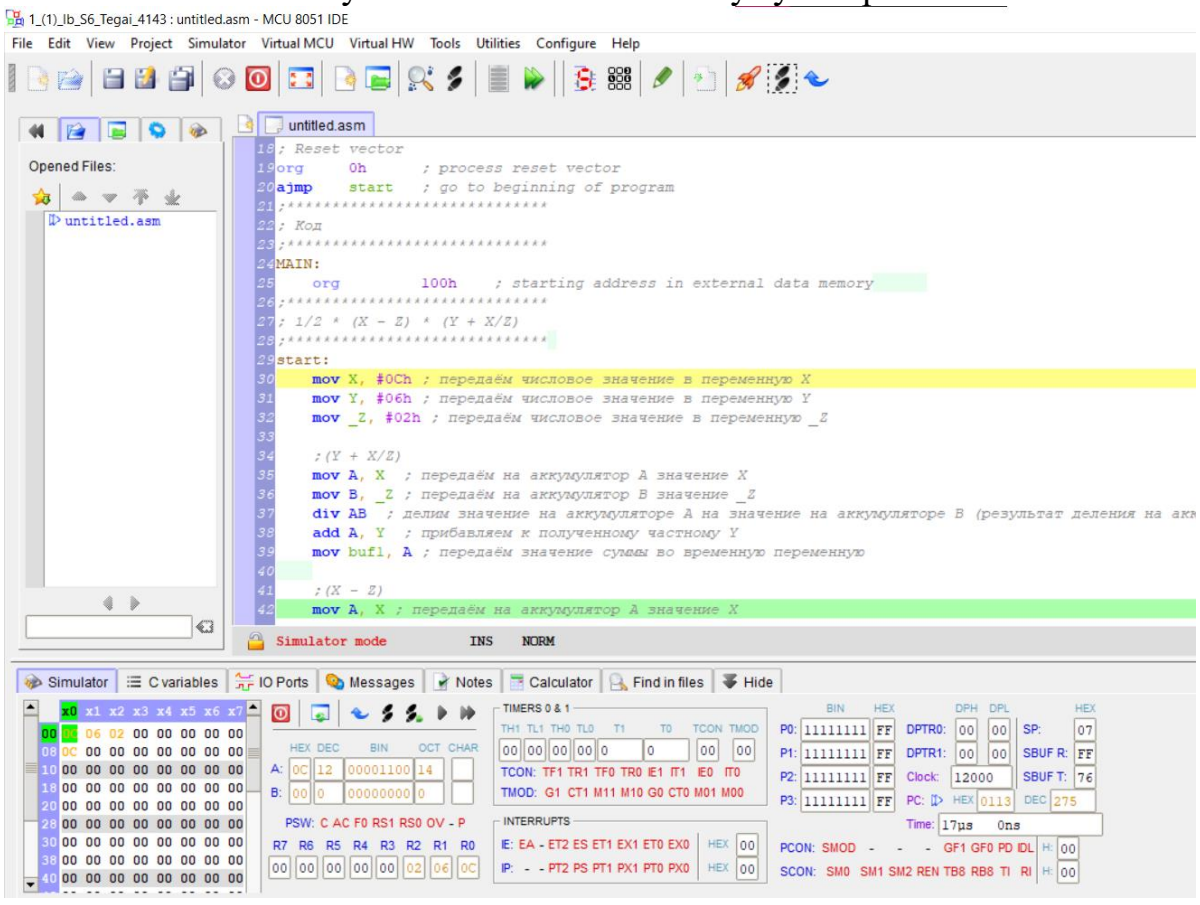

Рисунок 5 – Вывод первого промежуточного результата

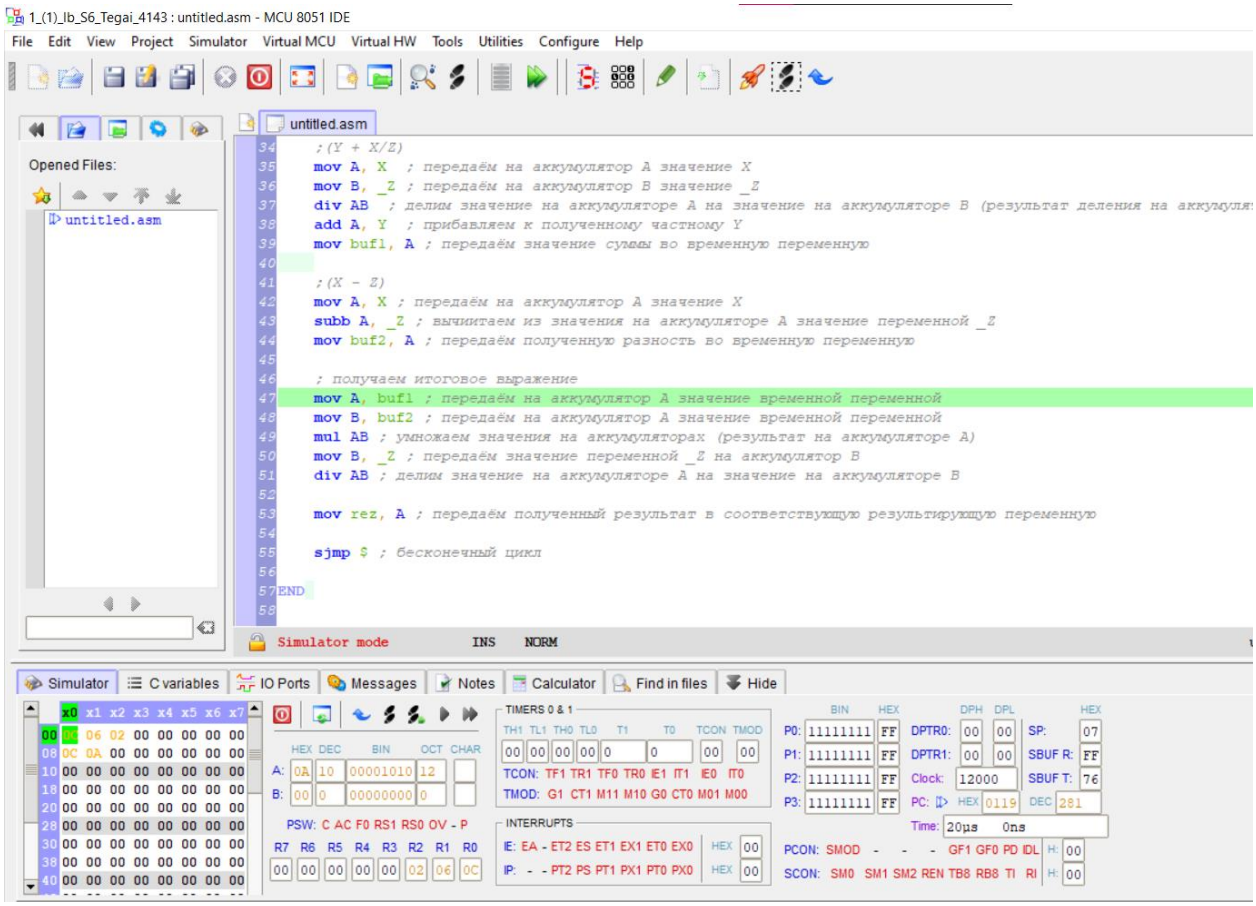

Рисунок 6 – Вывод второго промежуточного результата

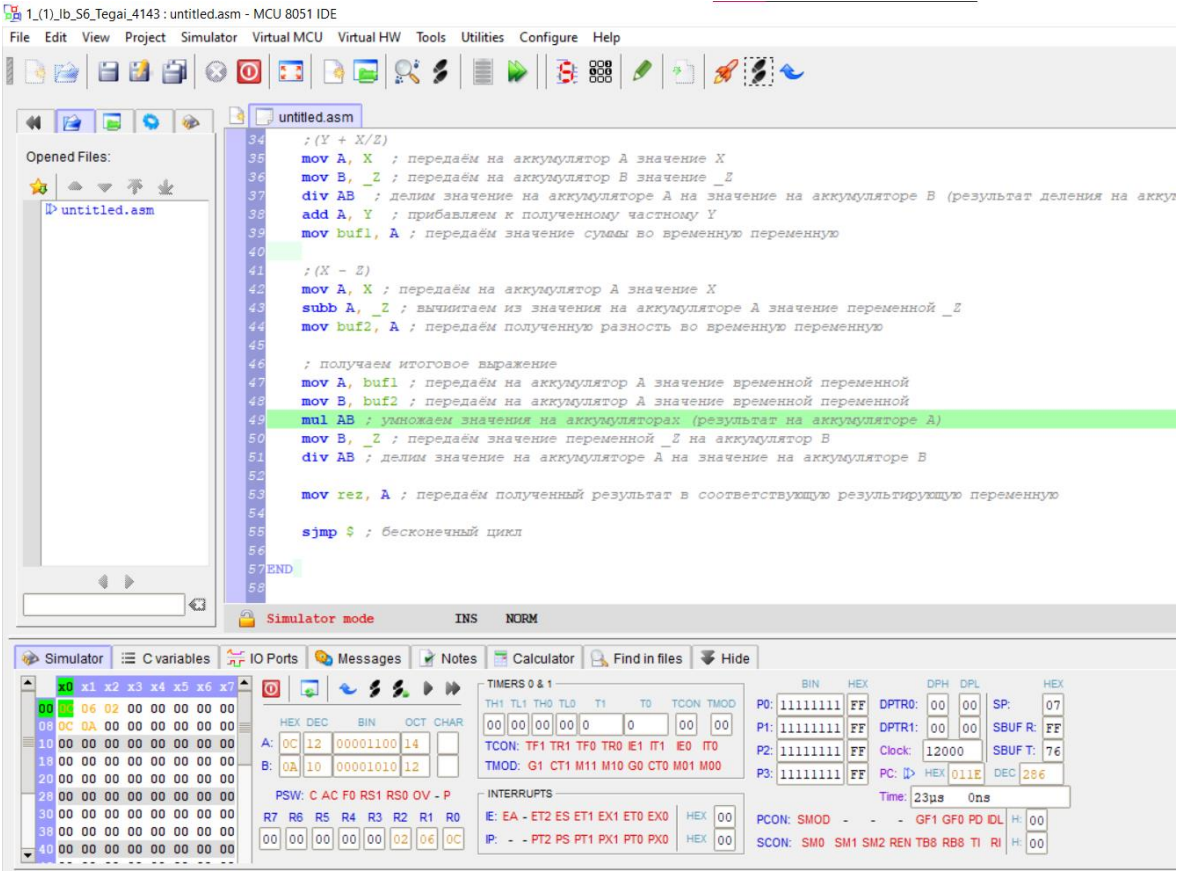

Рисунок 7 – Заполнение аккумуляторов

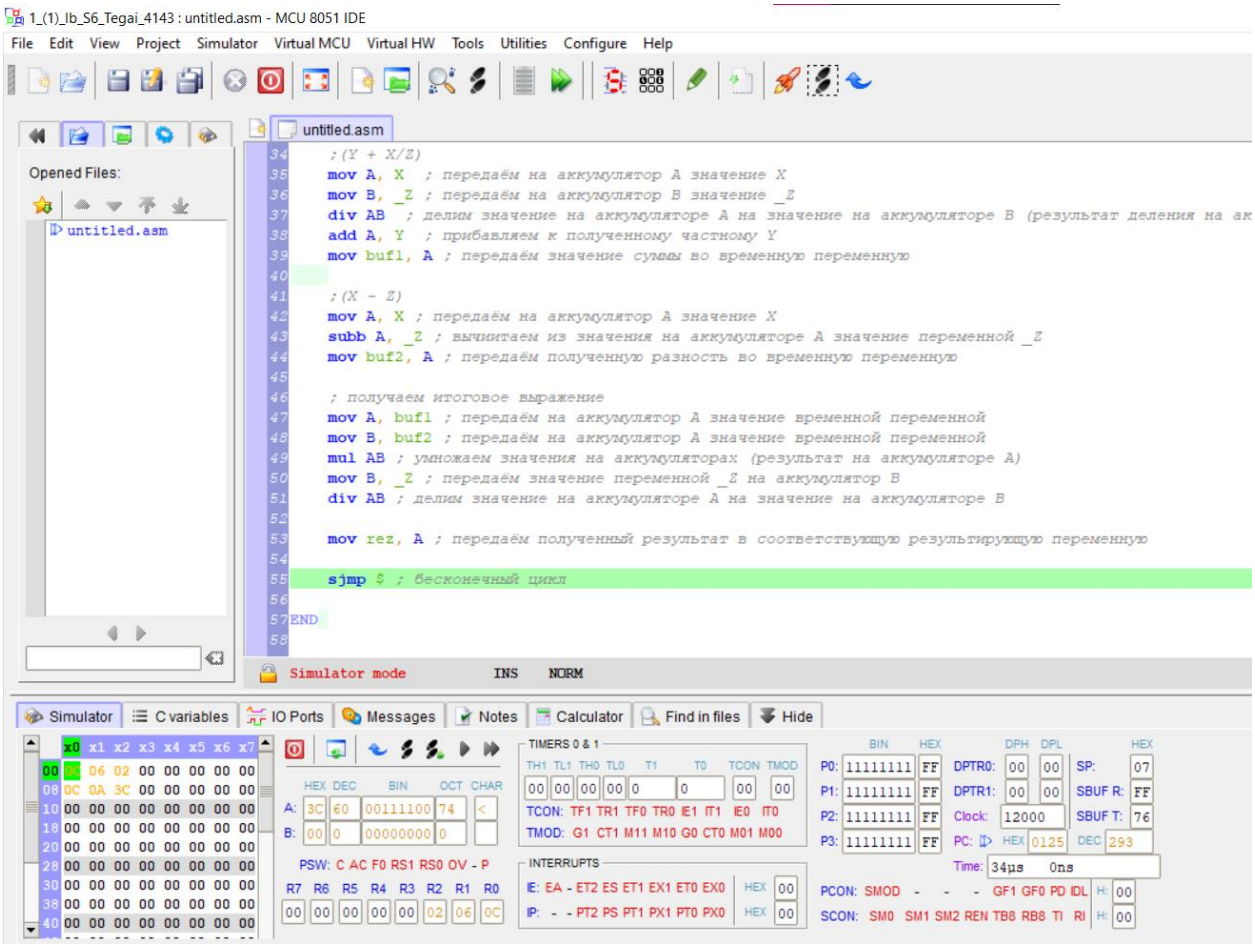

Рисунок 9 – Вывод итогового результата

Разработка программы 2

Логика алгоритма так же довольно проста. Грубо говоря, в массиве можно выделить 2 «граничных» значения – 0h и 15h. Если присмотреться, возвращение в 0h проводится ровно 2 раза, так что при разработке кода необходимо было воспользоваться цикличностью. Так как в ассемблере не существует таких понятий для цикла, как for и while, добиться цикличности можно с помощью счётчика. В качестве счётчика был выбран регистр R1, тогда как R0 – регистр, который хранил бы в себе первое значение массива (в дальнейшем – текущее, которое бы передавалось в R0 от аккумулятора A).

Можно выделить 2 части: инкрементирование и декрементирование элементов массива относительно граничащих (0h и 15h), так что были созданы метки, отвечающие за возрастание и убывание соответственно. Как только счётчик достигал определённого значения, осуществлялся переход на другую

метку, а иначе – повторная итерация по текущей метке. Это было достигнуто с помощью команды ***cjne***.

Самым сложным в разработке оказалось придумать код так, чтобы возвращений в 0h было ровно 2, а иначе код работал бесконечно. Для этого в код была добавлена метка проверки, которая бы проверяла значение счётчика, после совпадения с которым код переходил на метку завершения, а иначе – повторение итерации предыдущей метки.

Текст программы

```
;*******************************
;Filename: lb_1_S6_Tegai_4143
;Date: 2024/02/05
;File Version: 1
;Author: Tegai E.D.
;Company: SUAI
;Description: lb_1
;*******************************
;Reset Vector
;*******************************
org     0h      ; process reset vector
ajmp  start   ; go to beginning of program
;*******************************
;MAIN PROGRAM
;*******************************
org     100h  ; starting address in external data memory
start:
        mov R0,#0h ; инициализируем значение первого элемента массива
        mov A, #0h ; инициализируем значение аккумулятора (текущий элемент массива)
        mov R1, #0h ; инициализируем значение регистра R1 (счётчик итераций)
```

; метка для возрастания
massiv_vozrastanie:
movx @R0,A ; записываем текущее значение аккумулятора в память по адресу R0
inc R1 ; инкрементируем регистр R1
inc R0 ; инкрементируем регистр R0
inc A ; увеличиваем значение аккумулятора
cjne A, #0Fh, massiv_vozrastanie ; если значение 0, то повторяем возрастание, иначе переходим на убывание

; метка для убывания
massiv_ubivanie:
movx @R0,A ; записываем текущее значение аккумулятора в память по адресу R0
inc R1 ; инкрементируем регистр R1
dec R0 ; декрементируем регистр R0
dec A ; декрементируем значение аккумулятора
jnz massiv_ubivanie ; пока не 0, повторяем убывание
sjmp proverka ; переход на метку проверки (сделано для того, чтобы возвращение к 0 было ровно 2 раза)

; метка проверки
proverka:
cjne R1, #3Ch, massiv_vozrastanie ; пока счётчик не будет равняться значению #3Ch, идём на возрастание, иначе - в конец
sjmp done ; переход на метку конец

done:
sjmp $; бесконечный цикл

END

Результат работы программы

Результаты работы программы продемонстрированы на рисунках 10 - 15.

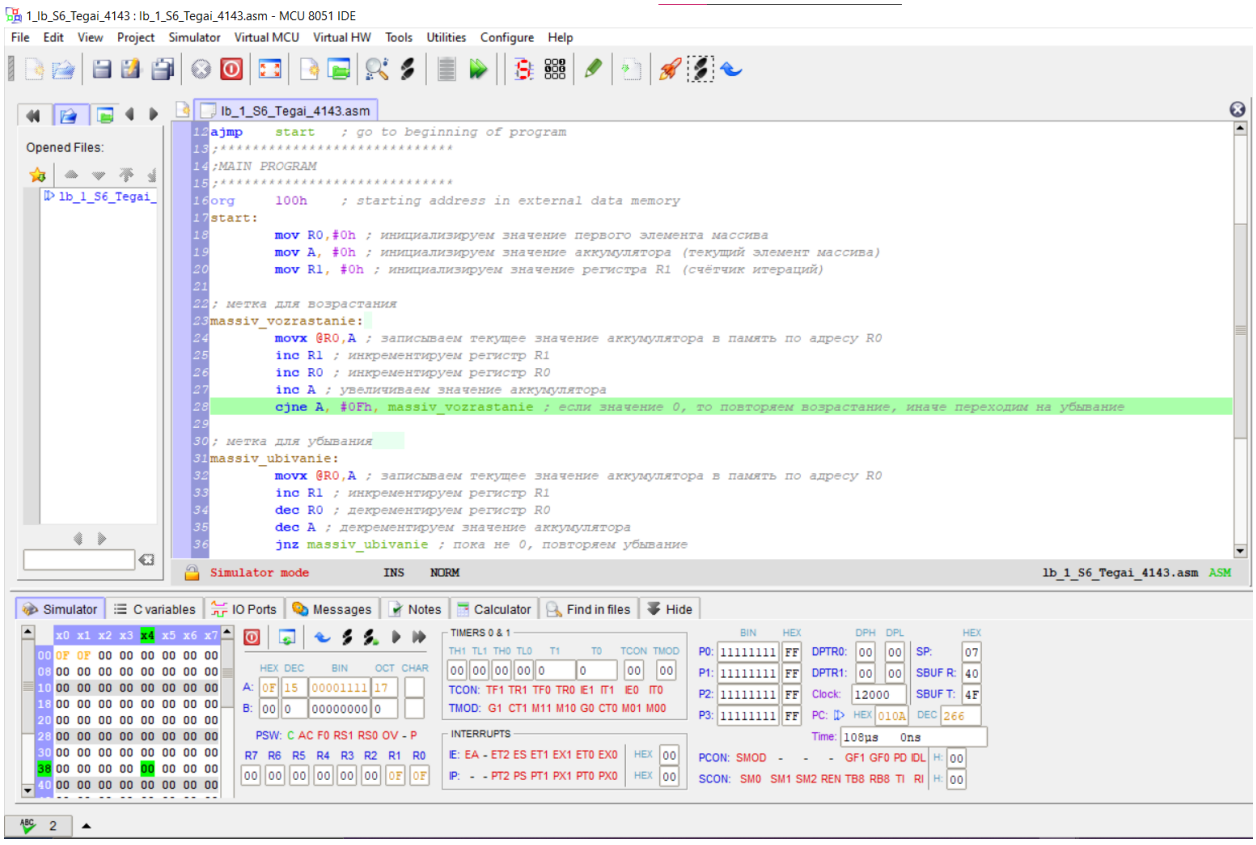

Рисунок 10 – Результат инкрементирования до 0F

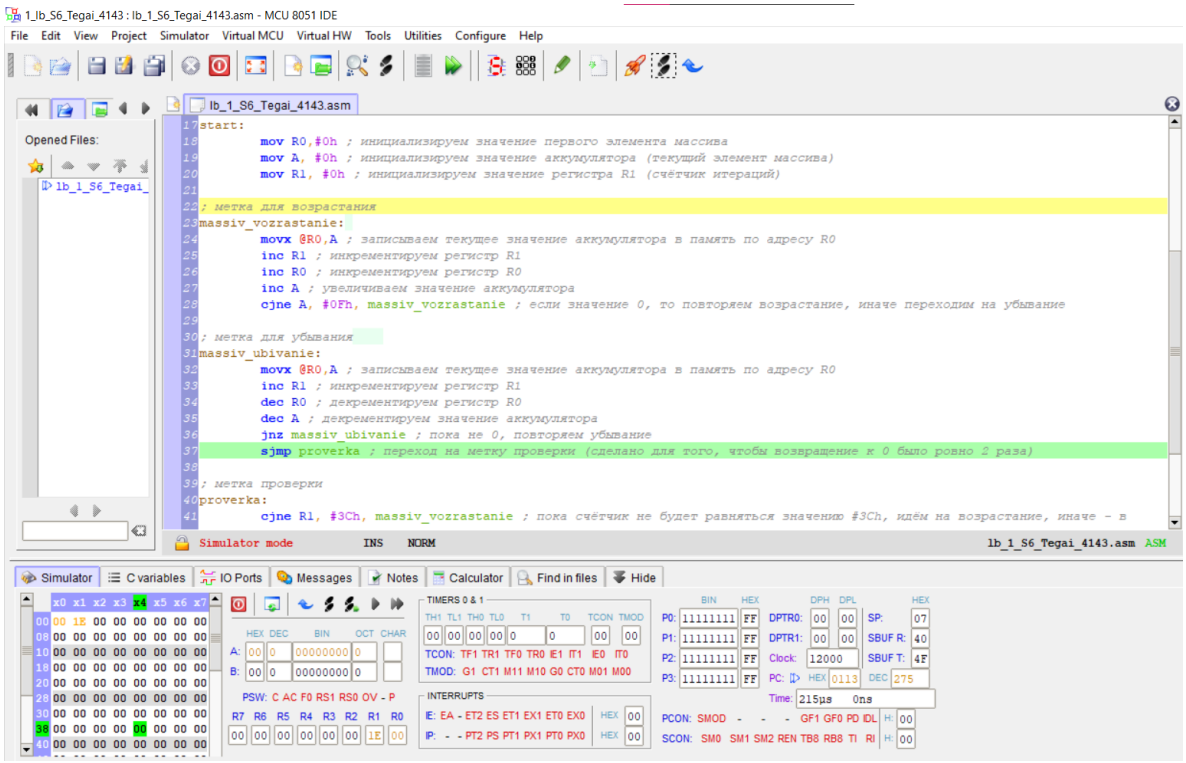

Рисунок 11 – Первое возвращение в 0

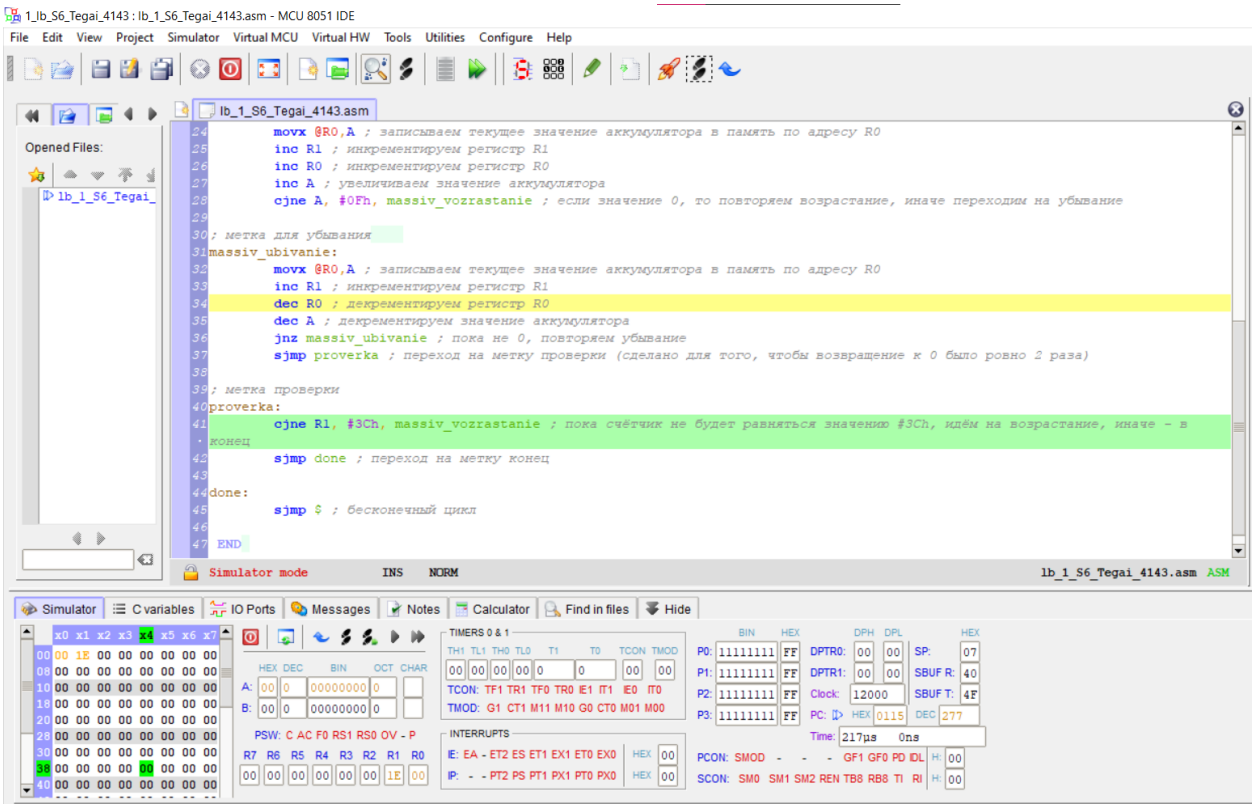

Рисунок 12 – Проверка на окончание цикла

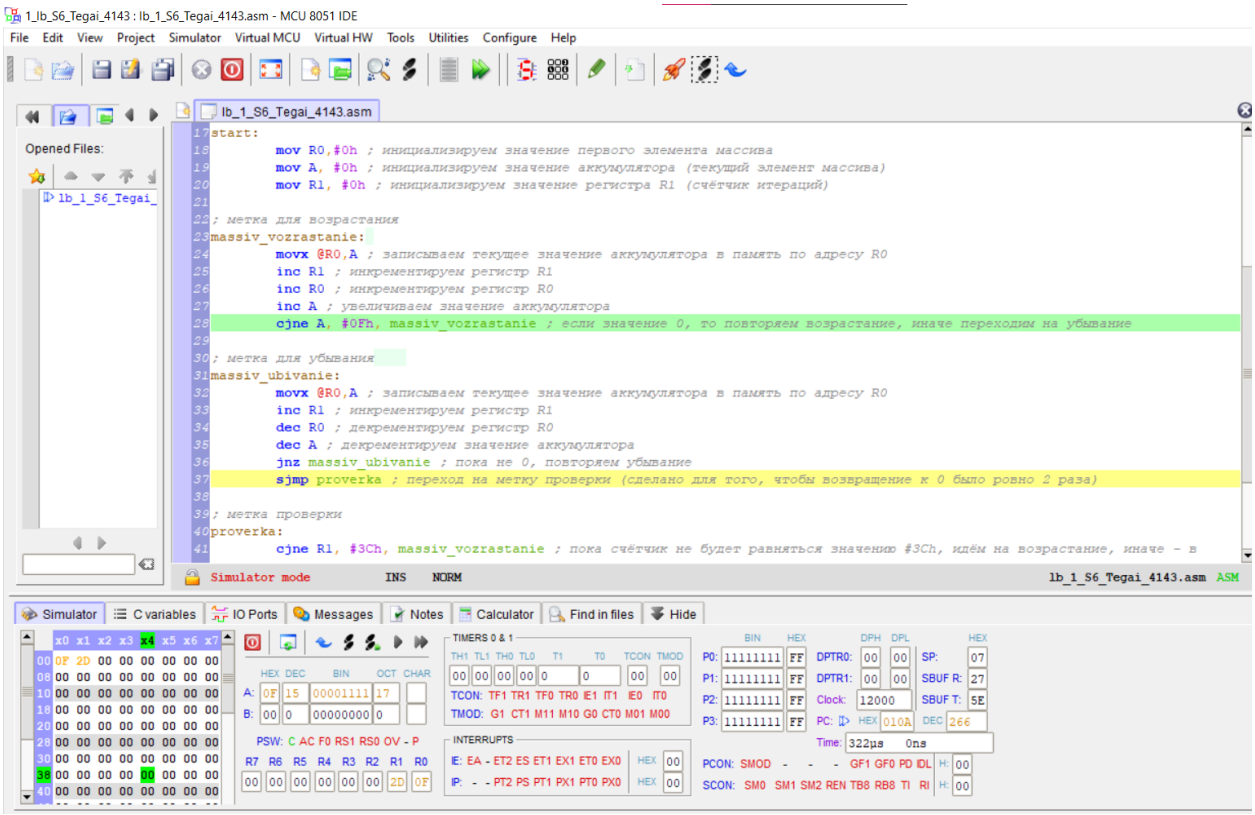

Рисунок 13 – Второе итерирование до 0F

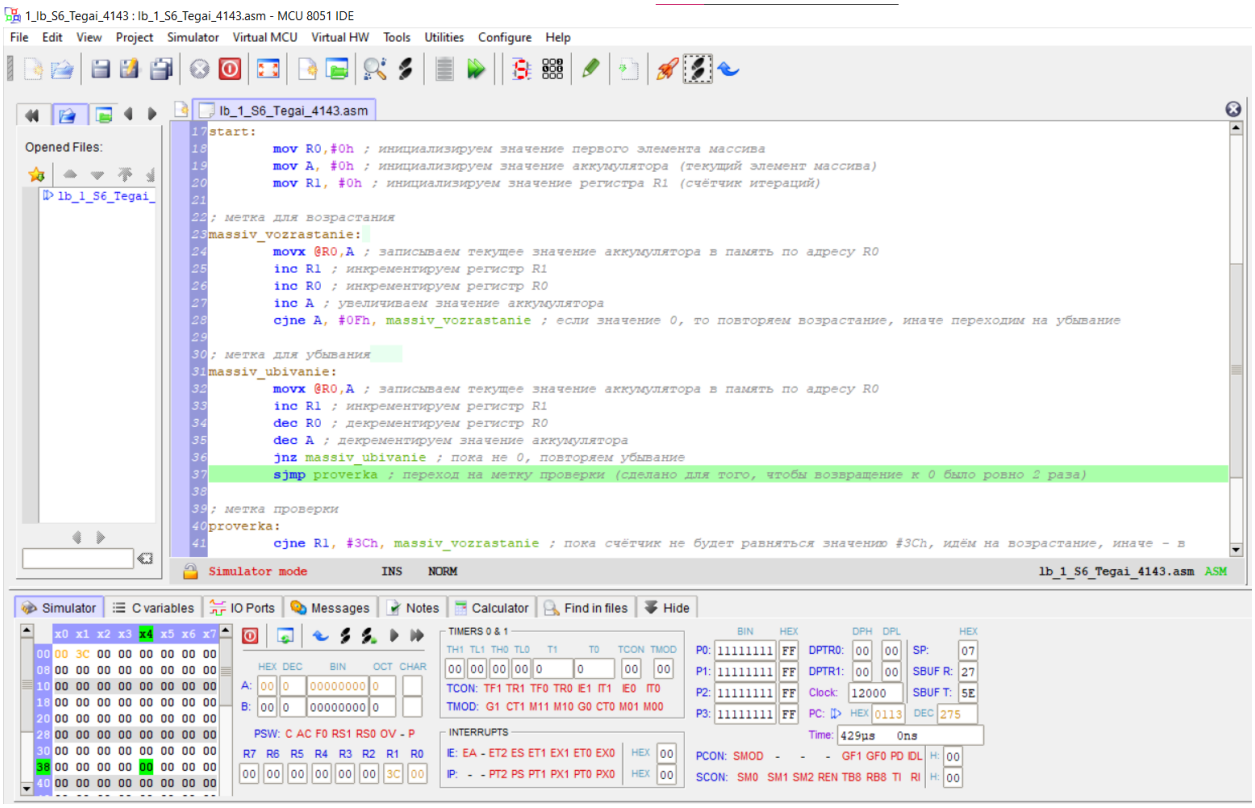

Рисунок 14 – Второе возвращение в 0

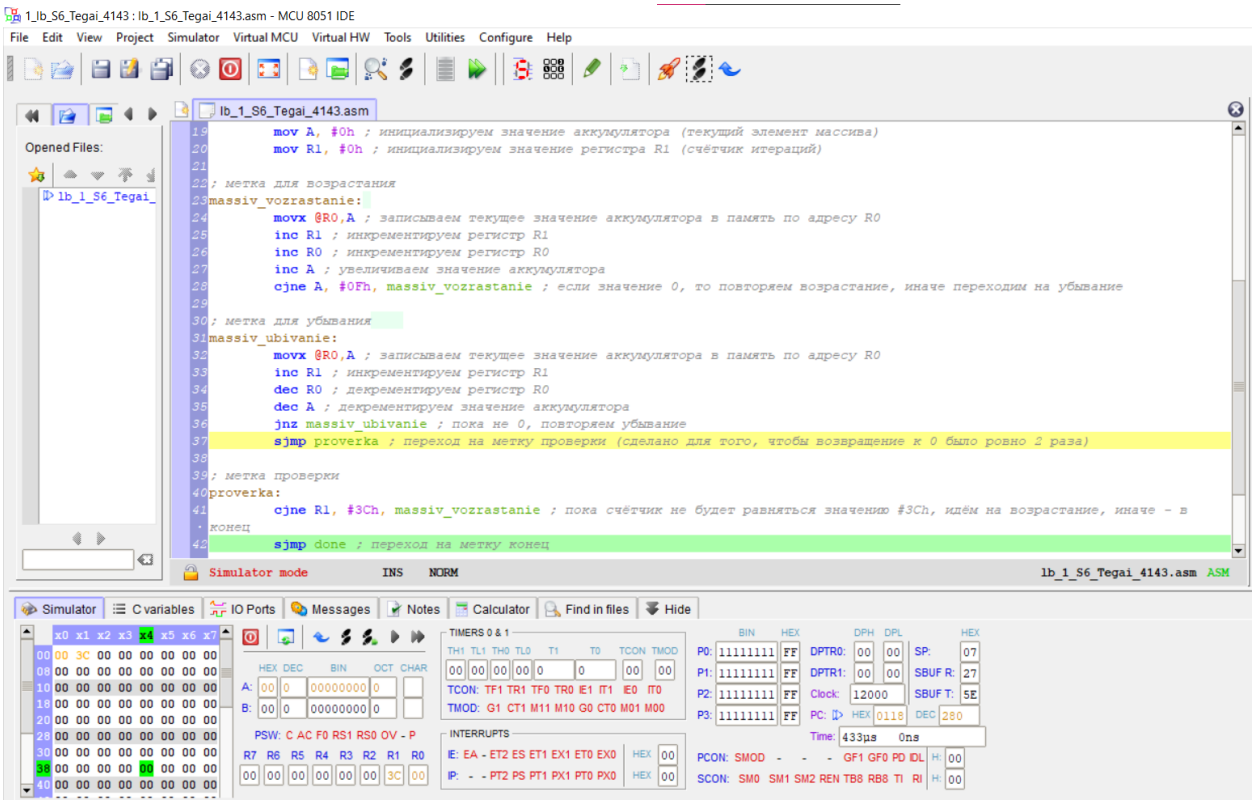

Рисунок 15 – Конец работы программы

Разработка программы 3

Разработка алгоритма программы для последнего задания по сложности ничем не отличается от разработки первого. Так что для начала нужно раскрыть сложение по модулю 2. Это показано на рисунке 16.

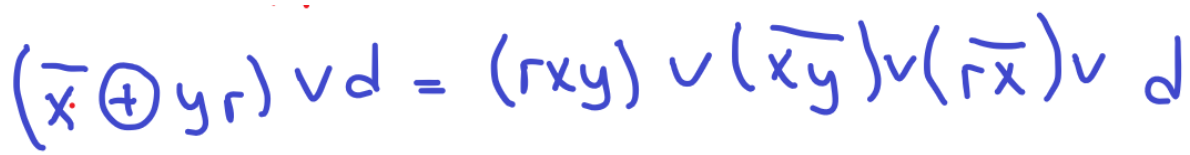

Рисунок 16 – Раскрытие сложения по модулю 2

Именно с этим выражением и будет связана логика работы кода. Порядок вычислений продемонстрирован на рисунке 17.

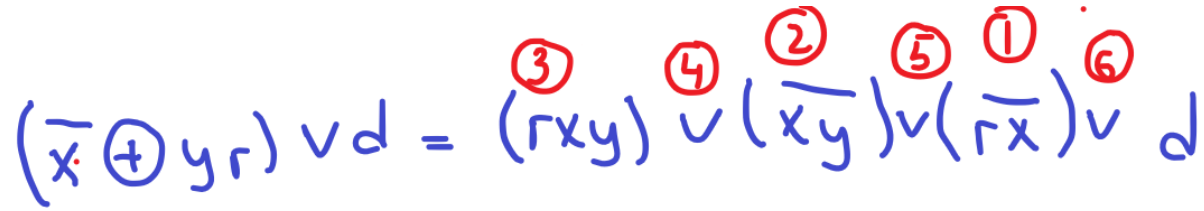

Рисунок 17 – Порядок вычислений

Для начала определяется место для хранения переменных x,y,r,d. Также определяются места хранения для промежуточных переменных buf1,buf2 и результирующей переменной rez (всё аналогично первому заданию, причина выбора именно такого количества промежуточных переменных не меняется).

Результатом работы является полученное значение на флаге C. Если он горит зелёным – результатом выражения является 1, иначе – 0.

Текст программы

```
;*******************************
;Filename: lb_1_S6_Tegai_4143
;Date: 2024/02/05
;File Version: 1
;Author: Tegai E.D.
;Company: SUAI
;Description: lb_1
;*******************************
; Переменные
;*******************************
```

```
x equ 18h ; задаём место хранения для переменной x
y equ 19h ; задаём место хранения для переменной y
r equ 1ah ; задаём место хранения для переменной r
d equ 1bh ; задаём место хранения для переменной d
buf1 equ 1ch ; задаём место хранения для переменной buf1, предназначенной для хранения промежуточных результатов
buf2 equ 1dh ; задаём место хранения для переменной buf2, предназначенной для хранения промежуточных результатов
rez equ 1eh ; задаём место хранения для результирующей переменной rez
;*****************************
;Reset Vector
;*****************************
RES_VECT CODE        0X0000 ; вектор сброса памяти в адресе 0X0000
sjmp start ; переход на метку начала программы
;*****************************
;MAIN PROGRAM
;*****************************
MAIN_PROG CODE 0x0100 ; метка основного кода, располагающегося в адресе 0x0100
; метка начала программ
;*****************************
; d V (/r/x) V (/x/y) V (rxy)
;*****************************
start:
        ;/r/x
        mov c,x ; передаём значение x на флаг c
        anl c,r ; умножаем значение флага c на r
        cpl c ; инвертируем значение флага c
        mov buf1, c ; передаём значение флага c во временную переменную
```

```
;/x/y
mov c,x ; передаём значение x на флаг c
anl c,y ; умножаем значение флага c на y
cpl c ; инвертируем значение флага c
mov buf2,c ; передаём значение флага c во временную переменную

;rxy
mov c,r ; передаём значение x на флаг c
anl c,x ; умножаем значение флага c на x
anl c,y ; умножаем значение флага c на y

;итоговое выражение
orl c,buf2 ; складываем второй и третий множители
orl c, buf1 ; складываем полученную сумму множителей на первый множитель
orl c, d ; складываем с d

mov rez,c ; передаём полученный результат в соответствующую результирующую переменную

sjmp $ ; бесконечный цикл

END
```

Результат работы программы

Результат работы программы продемонстрирован на рисунке 18.

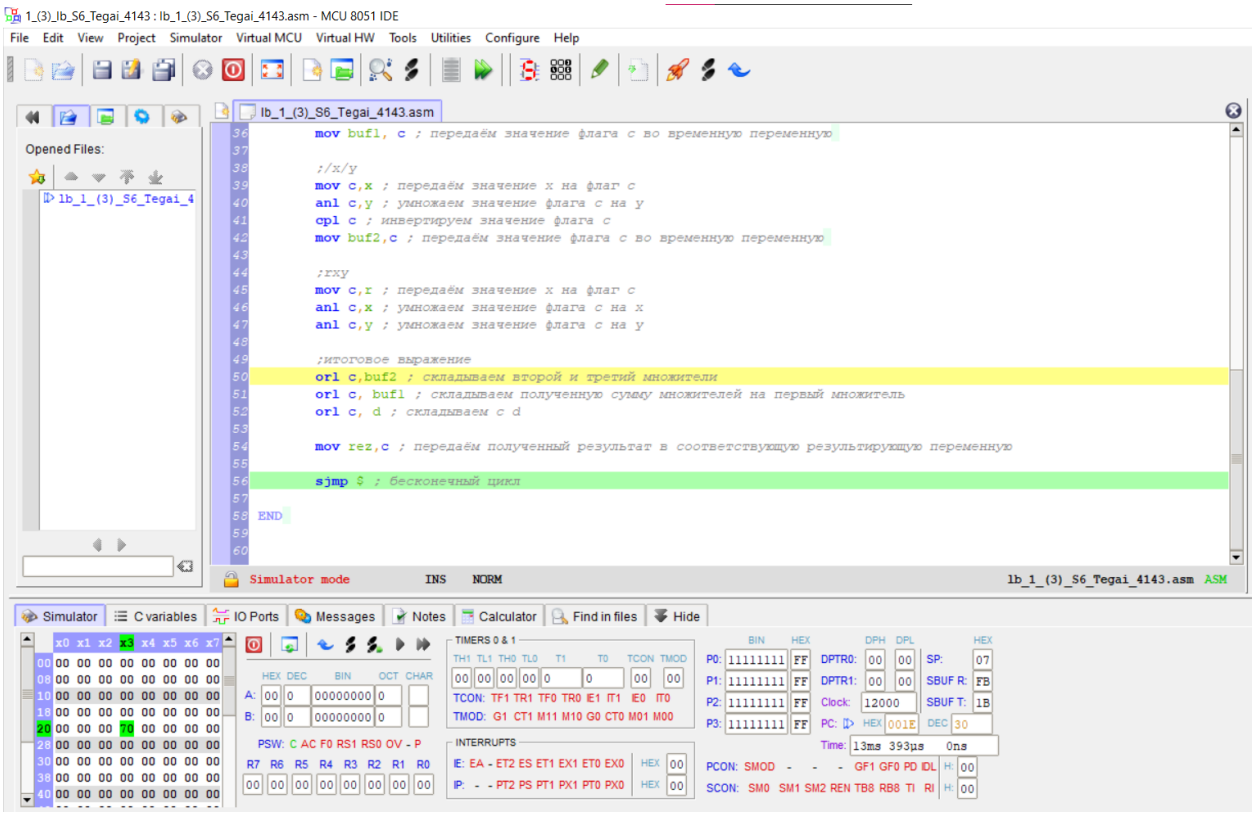

Рисунок 18 - Результат работы программы

Вывод

В результате выполнения работы созданы три программы на языке ассемблера MCS-51: программа для вычисления заданного арифметического выражения, программа для записи заданного массива чисел во внешнюю память данных, а также программа на ассемблере битового процессора для вычисления заданного логического выражения. Проверка работоспособности программ произведена в среде MCU 8051 IDE. Изучена архитектура и система команд микроконтроллера семейства MCS-51; приобретены навыки программирования микроконтроллера