МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ Государственное бюджетное образовательное учреждение высшего образования

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИиСП

Отчет

по лабораторной работе № 3

по дисциплине «Машинно-зависимые языки программирования»

Вариант 20

Выполнил: ст. гр. ПС-11

Ложкин С.А.

Проверил: доцент, доцент кафедры ИиСП Баев А.А.

г. Йошкар-Ола

2023

**Цель работы**: перевести код на C для микроконтроллера AVR в ассемблерный код

**Задания на лабораторную работу:**

1. По документации составить алгоритм перевода кода с C в ассемблерный
2. Перевести каждую команду
3. Полученный результат скомпоновать в готовую программу на ассемблере
4. **Теоретические сведения**

У нас есть код на С.

#include <avr/io.h>

#include <util/delay.h>

int main(void)

{

DDRC |= (1 << DDC5);

uint8\_t dir = 1;

while (1)

{

dir ^= 1;

if (dir)

{

PORTC |= (1 << PORTC5);

}

else

{

PORTC &= ~(1 << PORTC5);

}

\_delay\_ms(602);

}

return 0;

}

В документации можно посмотреть команды ассемблера, которые делают действия, аналогичные коду на C.

Так DDRC |= (1 << DDC5) совершается логическое или с регистром C по маске, то есть ставится 5 бит в единицу

Та же PORTC &= ~(1 << PORTC5) делает наоборот. Она выставляет бит в маске с помощью побитового сдвига, потом инвертирует эту маску и делает логическое и с PORTC. Получается, что в нём очистится 5 бит.

Задержка программы делается циклом с достаточно большим количеством итераций, чтобы нагрузить процессор вычислениями на определённое время

Мы должны уложиться в 602 мс, это 0,602 секунд.

Тактовая частота микроконтроллера = 16 мегагерц, то есть 16000000 тактов в секунду.

Умножаем задержку в секундах на количество тактов 0,602 \* 16000000 = 9632000. Это количество тактов, которое мы должны исполнить, чтобы получить задержку в 0,602 секунды.

1. **Практическая часть**

Сопоставляем каждой команде на C эквивалент на ассемблере

#include <avr/io.h>

#include <util/delay.h>

int main(void)

{

DDRC |= (1 << DDC5); // sbi DDRC,5 //Команда ставит конкретный бит по адресу регистра

uint8\_t dir = 1; // ldi r24,1 //Иницализируем переменную

while (1) // main\_loop: // Начало бесконечного цикла

{

dir ^= 1; // eor r24,r24 // искючающее или

if (dir) // brne set\_bit // Если не равно нулю, что ставим бит

{ // set\_bit:

PORTC |= (1 << PORTC5); //sbi 0x08,5 // ставим 5 бит

}

else

{

PORTC &= ~(1 << PORTC5); // cbi 0x08,5 // очищаем 5 бит

} // rjmp delay

// delay:

\_delay\_ms(602); // реализуем задержку

} // rjmp main\_loop // переходим в начало, чтобы получился бесконечный цикл

return 0;

}

Напишем реализацию задержки. Уже знаем, что надо заполнить процессором вычислениями на 9632000 такта.

Будем использовать вычитания из трёх регистров, два из которых ещё вычитают флаг переполнения, пока оба не станут равны 0.

У нас есть 3 загрузки в регистры (для первоначальных значений) по такту — всего 3 такта.

9632000 - 3 = 9631997. Столько тактов будет внутри цикла, пока третий регистр не переполнится.

За итерацию проходят 3 вычитания и условный переход — 5 тактов и минус 1 в самом конце, так как там нет перехода

Пусть в 17 регистре 255 , 18 — 255, а 19 — 10.

На 255 итерации:

r17 = 255 (переполнение)

r18 = 255 — 0 — 1(carry флаг)

r19 = 10 — 0 — 0 (carry флаг сбросился)

На 256 + 25 итерации:

r17 = 255 (переполнение)

r18 = 0 — 0 — 1(carry флаг) (переполнение)

r19 = 10 — 0 — 1 (carry флаг)

Таким образом, получается 5 \* (256 \* (256 \* 10 + 255) + 255) – 1 = 3604474

Этих чисел недостаточно, поэтому подбираем максимально оптимальный вариант. Начинаем с поиска значений для r20, потом подгоняем r19, потом r18. Это будет r17 = 0, r18 = 101, r19 = 29.

5 \* (256 \* (256 \* 29 + 101) + 0) – 1 = 9631999

ldi r17,0

ldi r18,101

ldi r19,29

delay\_loop:

subi r17,1

sbci r18,0

sbci r19,0

brne .delay\_loop; Если r19 не переполнилось, продолжаем цикл

Собираем всё в общую программу:

.global main

main:

sbi DDRC,5

ldi r24,1

main\_loop:

eor r24,r24

breq clear\_bit

sbi PORTC,5

rjmp delay

clear\_bit:

cbi PORTC,5

delay:

ldi r17,0

ldi r18,101

ldi r19,29

delay\_loop:

subi r17,1

sbci r18,0

sbci r19,0

brne .delay\_loop

rjmp main\_loop

**Выводы**

В ходе выполнения данной лабораторной работы я отлично понял, как пишутся программы на ассемблере для микроконтроллера AVR, и теперь могу применять эти знания на практике.