УНИВЕРСИТЕТ ИТМО

ФГАОУ ВО «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Факультет программной инженерии и компьютерной техники

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ

Лабораторная работа №2

Ввод-вывод численных данных

Нестеров Дали
Лабушев Тимофей
Группа P3302

Санкт-Петербург
2019

Цель работы

Познакомиться с двоично-десятичным и двоичным представлением целых и дробных чисел.

Совместить перевод из 10 в 2 и из 2 в 10 в одной программе для целых и дробных чисел и разработать программы на C51 и в Ассемблере A51 для ввода и вывода двузначных чисел. Сравнить листинги .lst программ в C51 и A51 и пояснить различия в программах.

Исходный текст программы на C51

```
#include <reg51.h>

void bcd_to_bin_int() {
        /* P1 contains a BCD */
        unsigned char bin, bcd;
        bin = (P1 >> 4) * 10 + (P1 & 0x0f);
        P2 = bin;

        bcd = (((bin / 10) % 10) << 4) | (bin % 10);
        P3 = bcd;
}

void bcd_to_bin_fixed_point() {
        /* P1 contains a BCD, convert it to binary bin_fixp = bin_int*2^n/10^m, n = 8, m = 2 */
        unsigned int bin;
        unsigned char bcd;
        bin = (P1 >> 4) * 10 + (P1 & 0x0f); /* binary integer */
        bin <<= 8; /* (*2^8) */
        bin = (bin % 100 > 50) ? bin / 100 + 1 : bin / 100; /* (/10^2 + rounding) */
        P2 = bin;

        bin *= 10; /* 0.ab -> a.b0 */
        bcd = (bin & 0x0f00) >> 4; /* write a to the first place (0.a_) */
        bcd |= (((bin & 0xff) * 10) & 0xf00) >> 8; /* a.b0 -> b.00, write b to the second place (0.ab) */
        P3 = bcd;
}

int main() {
        if (P0 == 0) bcd_to_bin_int();
        else bcd_to_bin_fixed_point();
        return 0;
}
```

Исходный текст программы на A51

```
cseg at 0

mov a, P0
jz handle_int
jmp handle_fixp

handle_int:
        ; bin = (P1 >> 4) * 10 + (P1 & 0x0f)
        mov a, P1
        lcall bcd_to_bin_int
        mov P2, a

        ; bcd = (((bin / 10) % 10) << 4) | (bin % 10)
        mov b, #10
        div ab ; a <- quotient(bin / 10), b <- remainder(bin % 10)
        mov r0, b ; r0 <- bin % 10
        mov b, #10
        div ab ; b <- (bin / 10) % 10
```

```
        mov a, b
        swap a ; << 4
        orl a, r0
        mov P3, a
        jmp terminate

handle_fixp:
        ; bin = (P1 >> 4) * 10 + (P1 & 0x0f);
        mov a, P1
        lcall bcd_to_bin_int
        ; bin <<= 8; /* (*2^8) */
        mov r1, a
        mov r0, #0
        mov r3, #0
        mov r2, #100
        lcall div16 ; r2 <- bin / 100, r0 <- bin % 100
        ; bin = (bin % 100 > 50) ? bin / 100 + 1 : bin / 100; /* (/10^2 + rounding) */
        mov a, r0
        clr c
        subb a, #50 ; carry = (bin % 100 > 50) ? 0 : 1
        jc  no_rounding
        inc r2
        no_rounding:
        mov P2, r2

        ; bin *= 10; /* 0.ab -> a.b0 */
        mov a, r2
        mov b, #10
        mul ab
        ; bcd = (bin & 0x0f00) >> 4; /* write a to the first place (0.a_) */
        anl b, #00fh
        mov r0, b ; r0 <- bin & 0x0f00
        ; bcd |= (((bin & 0xff) * 10) & 0xf00) >> 8; /* a.b0 -> b.00, write b to the second place (0.ab) */
        mov b, #10
        mul ab ; higher order bytes in b, low-order in a
        anl b, #00fh
        mov a, r0
        swap a ; (bin & 0x0f00) >> 4
        orl a, b
        mov P3, a
        jmp terminate

bcd_to_bin_int:
        mov r0, a
        anl a, #0f0h
        swap a ; a >> 4
        mov b, #10
        mul ab ; a <- (a * b)[0..7]
        mov b, a ; b <- higher digit
        mov a, r0
        anl a, #00fh ; a <- lower digit
        add a, b ; a + b = bcd converted to binary
        ret

$include (div16.a51)

terminate:
        end
```

Сравнение листингов

Размер кода

C51: `Program Size: data=9.0 xdata=0 code=290`

A51: `Program Size: data=8.0 xdata=0 code=152`

Пояснение

В отличие от скомпилированного кода, написанный вручную ассемблерный код более эффективно использует регистры и выполняет операции. Например, код на C выполняет операцию деления три раза для следующей конструкции:

```
bin = (bin % 100 > 50) ? bin / 100 + 1 : bin / 100;
```

в то время как ассемблерный код использует результат (частное и остаток) одной операции. Это отражается не только на размере кода, но и на скорости выполнения, поскольку операция 16-битного деления не входит в набор команд 8051 и реализуется программно.

Вывод