Университет ИТМО

Факультет программной инженерии и компьютерной техники
Кафедра вычислительной техники

Лабораторная работа № 2 по дисциплине "Организация ЭВМ и систем"

Выполнили: Айтуганов Д. А.
Чебыкин И. Б.

Группа: P3301

Проверяющий: Скорубский В. И.

СПб, 2016

Цель работы

1. Обретение базовых навыков работы с MCS51 в среде Keil.
2. Изучение архитектуры MCS51.
3. Изучение C51 и A51.

Задание

Разработать программу ввода и вывода целых чисел со знаком в C51 и A51 через порты.

Разработать программу ввода и вывода дробных двоичных чисел со знаком в C51 и A51 через порты.

Исходный код

Целые числа

```
#include <reg51.h>
typedef unsigned char uint8_t;
typedef unsigned int uint16_t;

void bin_to_bcd(uint16_t x, uint8_t *x0, uint8_t *x1) {
  uint16_t result = 0;
  uint8_t sign = 0;
  if((int)x < 0) {
    sign = 0xD0;
    x *= -1;
  }else sign = 0xC0;

  *x1 =  (((x / 10) % 10) << 4) | (x % 10);
  x = x / 100;
  *x0 =  x | sign;
}

uint16_t bcd_to_bin(uint8_t x0, uint8_t x1) {
    int result = ( ((x1 >> 4)) * 10 + (x1 & 0x0f) );
    return (result + (x0 & 0x0f) * 100) * ((x0 & 0xD0) ? -1 : 1);
}

void main(){
  uint16_t num = bcd_to_bin(P0, P1);
  uint8_t p2, p3;
    bin_to_bcd(num, &p2, &p3);
    P2 = p2;
    P3 = p3;
  while(1);
}
```

Листинг 1: src/int.c

Дробные числа

```
#include <reg51.h>

unsigned long y;
unsigned long res1;

main(){


  y = (P0 & 0x0f) * 100 + ((P1 & 0xf0) >> 4) * 10 + (P1 & 0x0f);
  y = y << 16;
  y = y / 1000;
  if (P0 & 0x10) y *= -1;
```

```
  P2 = 0;
  if ( (long)y < 0 ) {
    y *= -1;
    P2 = 0xD0;
  }else P2 = 0xC0
  y = y * 10;
  P2 |= ((y & 0xf0000)) >> 16;
  y = (y & 0x0ffff) * 10;
  P3 = (y & 0xf0000) >> 12;
  P3 |= ((y & 0xffff) * 10) >> 16;

  while(1);
}
```

Листинг 2: src/float.c

Реализация на ассемблере

```
DIV_VARS SEGMENT DATA
RSEG DIV_VARS
DIVIDEND: DS 2H
DIVISOR: DS 1H
QUOTIENT: DS 2H
REMAINDER: DS 1H

dseg at 0x20
  x: ds 2

cseg at 0x00
  mov SP, #0x28
  jmp x_to_P2P3

P0P1_to_x:

  ;(P0 & 0x0f) * 100
  mov a, P0
  jnb ACC.4, skip
  setb 2Fh.0 ;if (P0 & 0x10) c = 1;
  skip:
  anl a, #0x0F
  mov b, #0x64
  mul ab
  mov r0, b
  mov r1, a
  ;(P1 >> 4) * 10
  mov a, P1
  anl a, #0xF0
  swap a
  mov b, #0x0a
  mul ab
  mov r2, a
  ;(P1 & 0x0f)
  mov a, P1
  anl a, #0x0f
  add a, r2
  add a, r1
  mov r1, a
  ;if (P0 & 0x10) x *= -1;
  jnb 2Fh.0, skip2
  mov a, r1
  cpl a
  add a, #0x01
  mov r1, a
  mov a, r0
  cpl a
  addc a, #0x00
  mov r0, a
  skip2:
  mov x, r0
  mov x + 1, r1
  jmp finish

x_to_P2P3:
  mov x, #0xFC
  mov x + 1, #0x39
```

```
  ;if (x & 0x8000) { x *= -1; P2 = 0xd0; }
  mov P2, #0xC0
  mov a, x
  jnb ACC.7, skip3
  mov a, x + 1
  cpl a
  add a, #0x01
  mov x + 1, a
  mov a, x
  cpl a
  addc a, #0x00
  mov x, a
  mov P2, #0xD0
  skip3:
  ;P2 |= x / 100;
  mov DIVIDEND, x + 1
  mov DIVIDEND + 1, x
  mov DIVISOR, #0x64
  call D16BY8
  mov a, P2
  orl a, QUOTIENT
  mov P2, a
  ;P3 = (((x % 100) / 10) << 4) | x % 10;
  mov DIVIDEND, REMAINDER
  mov DIVIDEND + 1, #0x00
  mov DIVISOR, #0x0A
  call D16BY8
  mov a, QUOTIENT
  anl a, #0x0F
  swap a
  orl a, REMAINDER
  mov P3, a
  jmp finish


D16BY8: CLR A
  CJNE  A,DIVISOR,OK

DIVIDE_BY_ZERO:
  SETB  OV
  RET

OK: MOV QUOTIENT,A
  MOV R4,#8
  MOV R5,DIVIDEND
  MOV R6,DIVIDEND+1
  MOV R7,A

  MOV A,R6
  MOV B,DIVISOR
  DIV AB
  MOV QUOTIENT+1,A
  MOV R6,B

TIMES_TWO:
  MOV A,R5
  RLC A
  MOV R5,A
  MOV A,R6
  RLC A
  MOV R6,A
  MOV A,R7
  RLC A
  MOV R7,A

COMPARE:
  CJNE  A,#0,DONE
  MOV A,R6
  CJNE  A,DIVISOR,DONE
  CJNE  R5,#0,DONE
DONE: CPL C

BUILD_QUOTIENT:
  MOV A,QUOTIENT
  RLC A
  MOV QUOTIENT,A
```

```
  JNB ACC.0,LOOP

SUBTRACT:
  MOV A,R6
  SUBB  A,DIVISOR
  MOV R6,A
  MOV A,R7
  SUBB  A,#0
  MOV R7,A

LOOP: DJNZ  R4,TIMES_TWO

  MOV A,DIVISOR
  MOV B,QUOTIENT
  MUL AB
  MOV B,A
  MOV A,DIVIDEND
  SUBB  A,B
  MOV REMAINDER,A
  CLR OV
  RET

  finish:
END
```

Листинг 3: src/int.asm

```
dseg at 0x20
  x: ds 4

cseg at 0x00
  mov SP, #0x27
  jmp dec_to_float

float_to_dec:

  ;(P0 & 0x0f) * 100
  mov a, P0
  anl a, #0x0F
  mov b, #0x64
  mul ab
  mov x + 2, b
  mov x + 3, a
  ;(P1 & 0xf0) >> 4) * 10
  mov a, P1
  anl a, #0xF0
  swap a
  mov b, #0x0A
  mul ab
  add a, x + 3
  mov x + 3, a
  ;(P1 & 0x0f)
  mov a, P1
  anl a, #0x0F
  add a, x + 3
  mov x + 3, a
  ;y = y << 16
  mov x, x + 2
  mov x + 1, x + 3
  clr a
  mov x + 2, a
  mov x + 3, a
  ;y = y / 1000
  mov r4, x
  mov r5, x + 1
  mov r6, x + 2
  mov r7, x + 3
  mov r3, #0x0A
  call ?fast_long_divide
  mov r3, #0x0A
  call ?fast_long_divide
  mov r3, #0x0A
  call ?fast_long_divide
  ;if (P0 & 0x10) y *= -1;
  mov a, P0
  jnb ACC.4, skip
```

```
  mov a, r7
  cpl a
  add a, #0x01
  mov x + 3, a
  mov a, r6
  cpl a
  addc a, #0x00
  mov x + 2, a
  mov a, r5
  cpl a
  addc a, #0x00
  mov x + 1, a
  mov a, r4
  cpl a
  addc a, #0x00
  mov x, a
  skip:
  jmp prog_end

dec_to_float:
  mov x, #0xFF
  mov x + 1, #0xFF
  mov x + 2, #0x08
  mov x + 3, #0x73

  ;if (y & 0x80000000) {
  ; y *= -1;
  ; P2 = 0xD0;
  ;}
  ;else P2 = 0xC0;
  mov P2, #0xC0
  mov a, x
  jnb ACC.7, skip1
  mov a, x + 3
  cpl a
  add a, #0x01
  mov x + 3, a
  mov a, x + 2
  cpl a
  addc a, #0x00
  mov x + 2, a
  mov a, x + 1
  cpl a
  addc a, #0x00
  mov x + 1, a
  mov a, x
  cpl a
  addc a, #0x00
  mov x, a
  mov P2, #0xD0
  skip1:
  ;y = y * 10;
  mov R6, x + 2
  mov R7, x + 3
  mov R5, #0x0a
  call MUL16_16 ; now y is r0, r1, r2, r3
  mov a, R1
  orl a, P2
  mov P2, a
  ;y = (y & 0x0ffff) * 10;
  mov a, r2
  mov r4, a
  mov a, r3
  mov r5, a
  mov r7, #0x0a
  mov r6, #0x00
  call MUL16_16; now y is r0, r1, r2, r3
  ;P3 = (y & 0xf0000) >> 12;
  mov a, r1
  swap a
  mov P3, a
  ;P3 |= ((y & 0xffff) * 10) >> 16;
  mov a, r2
  mov r4, a
  mov a, r3
  mov r5, a
```

```
  mov r7, #0x0a
  mov r6, #0x00
  call MUL16_16; now y is r0, r1, r2, r3
  mov a, r1
  orl a, P3
  mov P3, a
  jmp prog_end

MUL16_16:
  ;
  ; R4_R5 * R6_R7 = R0_R1_R2_R3
  ;
  ; Byte 4  Byte 3  Byte 2  Byte 1
  ;*          R6    R7
  ;         R4    R5
  ;=  R0    R1    R2    R3
  ;
  ;Multiply R5 by R7
  MOV A,R5 ;Move the R5 into the Accumulator
  MOV B,R7 ;Move R7 into B
  MUL AB   ;Multiply the two values
  MOV R2,B ;Move B (the high-byte) into R2
  MOV R3,A ;Move A (the low-byte) into R3

  ;Multiply R5 by R6
  MOV A,R5    ;Move R5 back into the Accumulator
  MOV B,R6    ;Move R6 into B
  MUL AB      ;Multiply the two values
  ADD A,R2    ;Add the low-byte into the value already in R2
  MOV R2,A    ;Move the resulting value back into R2
  MOV A,B     ;Move the high-byte into the accumulator
  ADDC A,#00h ;Add zero (plus the carry, if any)
  MOV R1,A    ;Move the resulting answer into R1
  MOV A,#00h  ;Load the accumulator with  zero
  ADDC A,#00h ;Add zero (plus the carry, if any)
  MOV R0,A    ;Move the resulting answer to R0.

  ;Multiply R4 by R7
  MOV A,R4   ;Move R4 into the Accumulator
  MOV B,R7   ;Move R7 into B
  MUL AB     ;Multiply the two values
  ADD A,R2   ;Add the low-byte into the value already in R2
  MOV R2,A   ;Move the resulting value back into R2
  MOV A,B    ;Move the high-byte into the accumulator
  ADDC A,R1  ;Add the current value of R1 (plus any carry)
  MOV R1,A   ;Move the resulting answer into R1.
  MOV A,#00h ;Load the accumulator with zero
  ADDC A,R0  ;Add the current value of R0 (plus any carry)
  MOV R0,A   ;Move the resulting answer to R1.

  ;Multiply R4 by R6
  MOV A,R4  ;Move R4 back into the Accumulator
  MOV B,R6  ;Move R6 into B
  MUL AB    ;Multiply the two values
  ADD A,R1  ;Add the low-byte into the value already in R1
  MOV R1,A  ;Move the resulting value back into R1
  MOV A,B   ;Move the high-byte into the accumulator
  ADDC A,R0 ;Add it to the value already in R0 (plus any carry)
  MOV R0,A  ;Move the resulting answer back to R0

  ;Return - answer is now in R0, R1, R2, and R3
  RET

?fast_long_divide:                    ;

    // dividend in:           R4,R5,R6,R7
    // divisor in:            R3 only 0..16
    // uses:                  R0, R1, R2, A, B and DPL
    // quotient returned in:  R4,R5,R6,R7
    // remainder returned in: R0,R1,R2,R3

  PUSH    DPL                   ;
                ;
  MOV     A,PSW                 ;Get the base address of
  ANL     A,#0x18               ; the current register bank.
  ORL     A,#0x04               ;Point to register R4.
```

```
  MOV     R0,A                    ;R0 is a pointer to R4.
                  ;
  MOV     DPL,#04                 ;load loop counter.
                  ;
  MOV     R2,#00                  ;Clear the running remainder.
                                        ;
?fast_long_divide_loop:                 ;
                                        ;
  MOV     A,@R0                   ;Get the MS nibble of the
  SWAP    A                       ; dividend into the accumulator.
  ANL     A,#0x0F                 ;
  ORL     A,R2                    ;
                  ;
  MOV     B,R3                    ;Divide MS nibble by the divisor.
  DIV     AB                      ;
                  ;
  SWAP    A                       ;Save partial result (which must be
  MOV     R1,A                    ; in the range 0...15) shifted 4 bits.
                  ;
  MOV     A,B                     ;Save remainder (which must be in
  SWAP    A                       ; the range 0...15) shifted 4 bits left.
  MOV     R2,A                    ;
                  ;
  MOV     A,@R0                   ;Get next nibble of the dividend into
  ANL     A,#0x0F                 ; the accumulator.
  ORL     A,R2                    ;Or in the remainder from previous
                  ; partial division.
  MOV     B,R3                    ;Divide by the divisor.
  DIV     AB                      ;
                  ;
  ORL     A,R1                    ;Or in the previously saved partial result.
  MOV     @R0,A                   ;Save the MSB of the quotient ready
                  ; to be returned.
  MOV     A,B                     ;Save remainder shifted 4 bits left.
  SWAP    A                       ;
  MOV     R2,A                    ;
                  ;
  INC     R0                      ;

  DJNZ    DPL,?fast_long_divide_loop

  CLR     A                       ;Sort out the registers so that
  MOV     R0,A                    ; the remainder ends up in
  MOV     R1,A                    ; R3 with the other registers
  XCH     A,R2                    ; (R0,R1,R2) being zero.
  MOV     R3,A                    ;
                  ;
  POP     DPL                     ;
                  ;
  RET


prog_end:
  nop
END
```

Листинг 4: src/float.asm