Национальный исследовательский университет информационных технологий, механики и оптики

Кафедра вычислительной техники
Организация ЭВМ

Курсовая работа
«Проектирование микроЭВМ»
Вариант 8

Студенты:
Куклина М.,
Кириллова А.,
гр. P3301
Преподаватель:
Скорубский В.И.

Санкт-Петербург, 2017

1. Цель работы

Целью курсового проекта является разработка микропрограммного управления и схемы ЭВМ с архитектурой CISC и системой команд микроЭВМ (микрокомпьютер, MCU) MCS51. Исходными данными являются программная модель на уровне Ассемблера, перечень команд, выполняемых схемой. Для функционального описания микропрограмм и моделирования могут быть использованы языки программирования, наиболее близким из которых является язык Си в системе BorlandC++.

2. Система команд

2.1. `dec @rj`

C	AC	F0	RS1	RS0	OV		P

Команда записывает в ячейку памяти, адрес которой указан в регистре R_i, значение, на единицу меньшее текущего.

Размер 1 байт

Число циклов 1

Кодирование

0	0	0	0	0	1	1	i

Алгоритм $(R_i) \leftarrow (R_i) - 1$

Пример `DEC @R0`

2.2. `dec a`

C	AC	F0	RS1	RS0	OV		**P**

Команда записывает в аккумулятор a значение, на единицу меньшее текущего.

Размер 1 байт

Число циклов 1

Кодирование

0	0	0	1	0	1	0	0

Алгоритм $A \leftarrow A - 1$

Пример `DEC A`

2.3. `orl c, bit`

C	AC	F0	RS1	RS0	OV		P

Команда считывает бит, адрес которого указан в операнде `bit`, и записывает в C результат логического сложения C с этим битом.

Размер 2 байт

Число циклов 2

Кодирование

0 1 1 1 0 0 1 0	bit

Алгоритм $C \leftarrow C \vee b$

Пример `ORL C, 22h`

2.4. `orl c, /bit`

C	AC	F0	RS1	RS0	OV		P

Команда считывает бит, адрес которого указан в операнде `bit`, и записывает в C результат логического сложения C с битом, инверсным данному.

Размер 2 байт

Число циклов 2

Кодирование

1 0 1 0 0 0 0 0	bit

Алгоритм $C \leftarrow C \vee \neg b$

Пример `ORL C, /22h`

2.5. `mov a, @rj`

C	AC	F0	RS1	RS0	OV		**P**

Команда записывает в аккумулятор a содержимое ячейки памяти, адрес которой указан в регистре R_i.

Размер 1 байт

Число циклов 1

Кодирование

1 1 1 0 0 1 1 i

Алгоритм $A \leftarrow (R_i)$

Пример `MOV A, @R1`

2.6. `mov a, ad`

C	AC	F0	RS1	RS0	OV		**P**

Команда записывает в аккумулятор a содержимое ячейки памяти по адресу `ad`.

Размер 2 байт

Число циклов 1

Кодирование

1 1 1 0 0 1 0 1	ad

Алгоритм $A \leftarrow (\text{ad})$

Пример `MOV A, P0`

2.7. `jb bit, rel`

C	AC	F0	RS1	RS0	OV		P

Команда считывает бит `bit` и, если он установлен, переходит к адресу, указанному во втором операнде.

Размер 3 байт

Число циклов 2

Кодирование

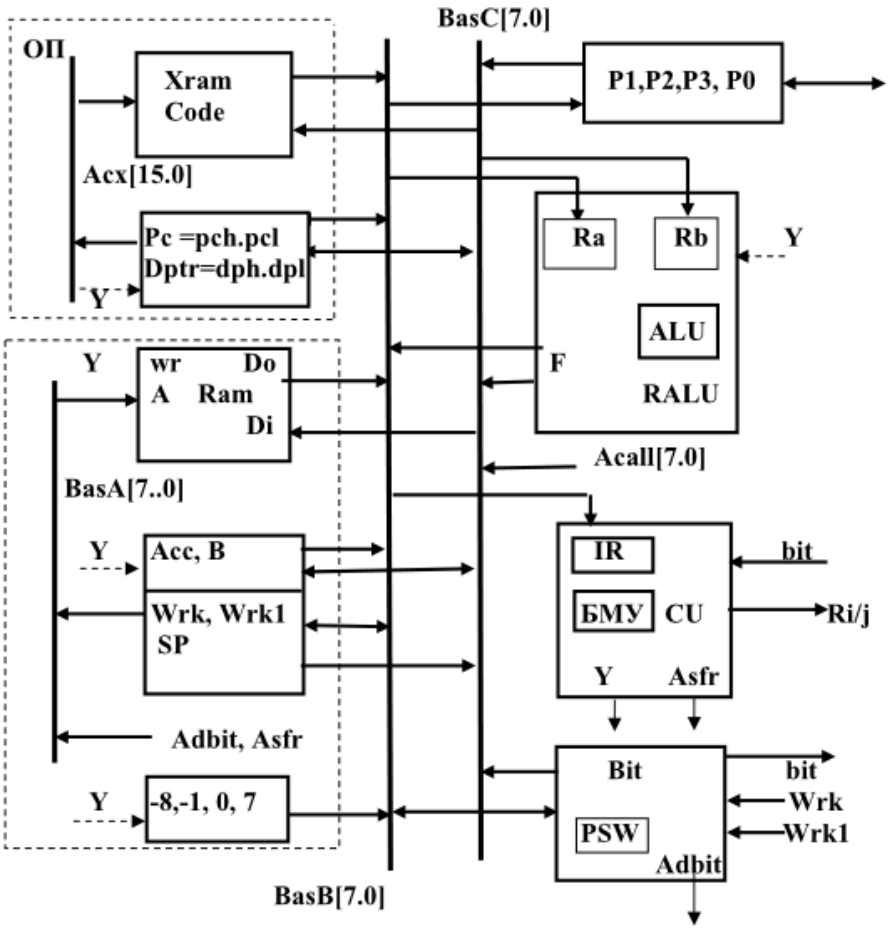
0 0 1 0 0 0 0 0	bit	rel

Алгоритм $\text{PC} \leftarrow \text{PC} + 3$
IF $(\text{bit}) = 1, \text{PC} \leftarrow \text{PC} + \text{rel}$

Пример `JB P1.2 LABEL`

---

3. Структура ЭВМ



4. Функциональное тестирование

```
cseg at 0x0
  ljmp start
cseg at 0x3
  reti

cseg at 0x4
  start:
      mov 0xAA, #0x04
    mov a, 0xAA
    dec a
    orl c, /ACC.1
    mov @R0, #0xAA
    dec @R0
    mov a, @R0
```

```
    orl c, ACC.1
    jb ACC.1, ok; test "jb TRUE, rel"
  err:
    ljmp err
  ok:
    jb ACC.7, err ; test "jb FALSE, rel"
  fin:
    ljmp fin
end
```

5. Сниппеты изменений

5.1. Формирование слова состояния PSWC

```
Q.printf("orlc ");
if (S == Q) {
    PSW |= (PB&(1 << (Wrk & 0x7))) ? 0x80 : 0x00;
}
```

5.2. Button1Click

Заполнение массива преобразования кода команды в адрес начала микрооперации.

```
// dec @ri  : j = 9;
for(uchar i = 0x16; i <= 0x17; ++i)
    ADC[i] = j;
++j;
// dec a : j = 10;
ADC[0x14] = j++;
// orl c, bit : j = 11;
ADC[0x72] = j++;
// orl c, /bit : j = 12;
ADC[0xA0] = j++;
// mov a, @rj : j = 13;
for(uchar i = 0xE6; i <= 0xE7; ++i)
    ADC[i] = j;
++j;
// mov ri, #d : j = 14;
for(uchar i = 0x76; i <= 0x77; ++i)
    ADC[i] = j;
++j;
// mov a, ad : j = 15;
ADC[0xE5] = j++;
// jb bit, rel : j = 16;
ADC[0x20] = j++;
// mov ad, #d : j = 17;
ADC[0x75] = j++;
```

Сброс и заполнение программной памяти.

```
for (i = 0; i < 100; i++)
      CODE[i] = 0; // Reset
    PC = 0;
    //00: ljump 0x06
    CODE[PC++] = 0x02;
    CODE[PC++] = 0x00;
    CODE[PC++] = 0x04;
    //03: reti
    CODE[PC++] = 0x32;
    //04: mov 0xaa, #0x04
    CODE[PC++] = 0x75;
    CODE[PC++] = 0xaa;
    CODE[PC++] = 0x04;
    //07: mov a, 0xaa
    CODE[PC++] = 0xE5;
    CODE[PC++] = 0xaa;
    //09: dec a
    CODE[PC++] = 0x14;
    //0A: orl c, /ACC.1
    CODE[PC++] = 0xA0;
    CODE[PC++] = 0xE1;
    //0C: mov @r0, #0xaa
    CODE[PC++] = 0x77;
    CODE[PC++] = 0xaa;
    //0E: dec @r0
    CODE[PC++] = 0x17;
    //0F: mov a, @r0
    CODE[PC++] = 0xE7;
    //10: orl c, ACC.1
    CODE[PC++] = 0x72;
    CODE[PC++] = 0xE1;
    //12: jb Acc.1, ok ; true condition
    CODE[PC++] = 0x20;
    CODE[PC++] = 0xE1;
    CODE[PC++] = 0x03;
    //15: ljmp err
    CODE[PC++] = 0x02;
    CODE[PC++] = 0x00;
    CODE[PC++] = 0x15;
    //1b: jb acc.4, err ; false condition
    CODE[PC++] = 0x20;
    CODE[PC++] = 0xE4;
    CODE[PC++] = 0xfa;
    //1E: ljmp fin
    CODE[PC++] = 0x02;
    CODE[PC++] = 0x00;
    CODE[PC++] = 0x1b;
```

5.3. Исполнение теста Button2OnClick

```
case 9: // dec @ri
{
  { // 9.1. Read Ri (can only be R0 or R1)
    Wrk = Ram[(PSW & 0x18) | (IR & 0x1)];
    ss[0] = (IR & 0x1) | 0x30;
    ss[1] = '\0';
    char code[10] = "dec @r";
    Instr->Text = StrCat(code, ss);
  }
  { // 9.2. Read @Ri
    PB = Ram[Wrk] - 1;
  }
  { // 9.3. Write [@Ri - 1] to @Ri
    Ram[Wrk] = PB;
  }
  goto finish;
}
case 10: // dec a
{
  { // 10.1. Modifying A
    Wrk = Ram[Acc] - 1;
  }
  { // 10.2. Writing A
    Ram[Acc] = ACC = Wrk;
    char code[12] = "dec a";
    ss[0] = '\0';
    Instr->Text = StrCat(code, ss);
  }
  { // 10.3. Updating the status register
    ss[0] = 'a';
    ss[1] = 'b';
    ss[2] = '\0';
    PSWC(ss);
    Ram[Psw] = PSW;
  }
  goto finish;
}
case 11: // orl c, bit
{
  { // Reading the bit address
    Wrk = CODE[PC++];
    RAMK++;
    // Wrk = Ex = 1110 xxxx
    ss[0] = (Wrk & 0x0F) | 0x30;
    ss[1] = '\0';
```

```
      char code[12] = "orl c, Acc.";
      Instr->Text = StrCat(code, ss);
    }
    { // 7.2. Reading from SFR or RAM
      if (Wrk & 0x80)
        PB = Ram[Wrk & 0xf8];
      else
        PB = Ram[0x20 | ((Wrk & 0x78) >> 3)];
      RAMK++;
    }
    { // 7.3.
      PSWC("orlc ");
      RAMK++;
    }
    {
      Ram[Psw] = PSW;
      RAMK = 0;
    }
    goto finish;
  }
  case 12: // orl c, /bit
  {
    { // Reading the bit address
      Wrk = CODE[PC++];
      ss[0] = (Wrk & 0x7) | 0x30;
      ss[1] = '\0';
      char code[12] = "orl c, /";
      Instr->Text = StrCat(code, ss);
      RAMK++;
    }
    { // 7.2. Reading from SFR or RAM
      if (Wrk & 0x80)
        PB = Ram[Wrk & 0xf8];
      else
        PB = Ram[0x20 | ((Wrk & 0x78) >> 3)];
      PB = ~PB;
      RAMK++;
    }
    { // 7.3.
      PSWC("orlc ");
      RAMK++;
    }
    {
      Ram[Psw] = PSW;
      RAMK = 0;
    }
    goto finish;
```

```
}
case 13: // mov a, @rj
{
  {
    Wrk = Ram[(PSW & 0x18) | (IR & 0x1)];
    RAMK++;
    ss[0] = (IR & 0x1) | 0x30;
    ss[1] = 0;
    char code[10] = "mov a,@r";
    Instr->Text = StrCat(code, ss);
  }
  {
    Ram[Acc] = ACC = Ram[Wrk];
    RAMK++;
    PSWC(ss);
  }
  {
    Ram[Psw] = PSW;
    RAMK = 0;
  }

  goto finish;
}
case 14: // mov ri, #d
{
  { // 14.1. Read Ri (can only be R0 or R1)
    /* Debug */ Wrk = CODE[PC++];
    Ram[(PSW & 0x18) | (IR & 0x1)] = Wrk;
    ss[0] = (IR & 0x1) | 0x30;
    ss[1] = '\0';
    char code[10] = "mov @r";
    Instr->Text = StrCat(code, ss);
  }
  goto finish;
}
case 15: // mov a, ad
{
  {
    Wrk = CODE[PC++];
    RAMK++;
    itoa(Wrk, ss, 16);
    char code[10] = "mov a, ";
    Instr->Text = StrCat(code, ss);
  }
  {
    ACC = Ram[Wrk];
    RAMK++;
```

```
      PSWC(ss);
    }
    { Ram[Acc] = ACC; }
    {
      Ram[Psw] = PSW;
      RAMK = 0;
    }

    goto finish;
  }
  case 16: // jb bit, rel
  {
    { // Reading the bit address
      Wrk = CODE[PC++];
    }
    {
      PA = CODE[PC++];
      RAMK++;
      ss[0] = (Wrk & 0x7) | 0x30;
      ss[1] = '\0';
      char code[12] = "jb Acc.";
      Instr->Text = StrCat(code, ss);
    }
    { // 7.2. Reading from SFR or RAM
      if (Wrk & 0x80)
        PB = Ram[Wrk & 0xF0];
      else
        PB = Ram[0x20 | ((Wrk & 0x78) >> 3)];
      RAMK++;
    }
    {
      char tmp = Wrk & 0x7;
      if (PB & (tmp & 0x1 ? tmp + 1 : tmp))
        PC += PA;
    }
    goto finish;
  }
  case 17: // mov ad, #d
  {
    {
      Wrk = CODE[PC++];
      RAMK++;
      itoa(Wrk, &ss[0], 16);
      char code[10] = "mov ad,#";
      Instr->Text = StrCat(code, ss);
    }
    {
```

```
            Ram[Wrk] = CODE[PC++];
            RAMK++;
        }
        goto finish;
    }
```

Вывод

В ходе выполнение курсовой работы изучалась архитектура micro51 и принципы проектирования и реализации ЭВМ. В результате выполнения работы были разработаны и интегрированы команды в соответствии с выданным вариантом.