Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

Организация ЭВМ и систем

Курсовая работа

Старался: Шкаруба Н.Е.

Группа: P3318

2017

Цель работы:

Разработка микропрограммного управления и схемы ЭВМ с архитектурой CISC и системой микрокоманд MCS51. Исходными данными является программная модель на уровне ассемблера, перечень команд, выполняемых схемой и элементная база MaxPlus.

Для функционального описания микропрограмм и моделирования могут быть использованы языки программирования, наиболее близким из которых является Си в системе BorlandC++. Схема проекта разрабатывается в системе MaxPlus и загружается в ПЛИС фирмы Алтера. Верификация проекта выполняется в симуляторе MaxPlus.
Для описания, визуального моделирования, кодирования и создания загрузочных файлов в проекте MaxPlus используется система BorlandC++.

Этап 0. Вариант.

8	**dec {@rj,a}**	**orl c, {bit,/bit}**	**mov a,{@rj,ad}**	**jb bit,rel**

Этап 1. Описание команд.

Dec

The DEC instruction decrements the specified operand by 1. An original value of 00h underflows to 0FFh. No flags are affected by this instruction.

DEC @Ri

C	AC	F0	RS1	RS0	OV		P

Bytes 1

Cycles 1

Encoding 0001011i

Operation
DEC
(Ri) = (Ri) - 1

Example DEC @R1

DEC A

C	AC	F0	RS1	RS0	OV		P

Bytes 1

Cycles 1

Encoding 00010100

Operation
DEC
A = A - 1

Example DEC A

DEC direct

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 1

Encoding 00010101 direct

Operation
DEC
(direct) = (direct) - 1

Example DEC 35h

ORL

The ORL instruction performs a bitwise logical OR operation on the specified operands, the result of which is stored in the destination operand.

ORL A, #immediate

C	AC	F0	RS1	RS0	OV		P

Bytes	2	
Cycles	1	
Encoding	01000100	immediate
Operation	ORL A = A OR immediate	
Example	ORL A, #01h	

ORL A, @Ri

C	AC	F0	RS1	RS0	OV		P

Bytes	1	
Cycles	1	
Encoding	0100011i	
Operation	ORL A = A OR (Ri)	
Example	ORL A, @R0	

ORL A, direct

C	AC	F0	RS1	RS0	OV		P

Bytes	2	
Cycles	1	
Encoding	01000101	direct
Operation	ORL A = A OR (direct)	
Example	ORL A, P0	

ORL A, Rn

C	AC	F0	RS1	RS0	OV		P

Bytes 1

Cycles 1

Encoding 01001nnn

Operation ORL
A = A OR Rn

Example ORL A, R5

ORL C, /bit

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 2

Encoding 10100000 bit

Operation ORL
C = C OR NOT (bit)

Example ORL C, /22h

ORL C, bit

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 2

Encoding 01110010 bit

Operation ORL
C = C OR (bit)

Example ORL C, 22h

ORL direct, #immediate

C	AC	F0	RS1	RS0	OV		P

Bytes 3

Cycles 2

Encoding 01000011 direct immediate

Operation ORL
(direct) = (direct) OR immediate

Example ORL P0, #01h

ORL direct, A

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 1

Encoding 01000010 direct

Operation ORL
(direct) = (direct) OR A

Example ORL P0, A

MOV

The MOV instruction moves data bytes between the two specified operands. The byte specified by the second operand is copied to the location specified by the first operand. The source data byte is not affected.

MOV @Rn, #immediate

C	AC	F0	RS1	RS0	OV		P

Bytes	2	
Cycles	1	
Encoding	0111011n	immediate
Operation	MOV (Rn) = immediate	
Example	MOV @R0, #0	

MOV @Ri, A

C	AC	F0	RS1	RS0	OV		P

Bytes	1
Cycles	1
Encoding	1111011i
Operation	MOV (Ri) = A
Example	MOV @R0, A

MOV @Ri, direct

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 2

Encoding 1010011i direct

Operation MOV
(Ri) = (direct)

Example MOV @R1, P2

MOV A, #immediate

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 1

Encoding 01110100 immediate

Operation MOV
A = immediate

Example MOV A, #0FFh

MOV A, @Ri

C	AC	F0	RS1	RS0	OV		P

Bytes 1

Cycles 1

Encoding 1110011i

Operation MOV
A = (Ri)

Example MOV A, @R1

MOV A, direct

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 1

Encoding 11100101 direct

Operation MOV
A = (direct)

Example MOV A, P0

MOV bit, C

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 2

Encoding 10010010 bit

Operation MOV
(bit) = C

Example MOV 22h, C

MOV C, bit

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 1

Encoding 10100010 bit

Operation MOV
C = (bit)

Example MOV C, 22h

MOV dest_direct, src_direct

C	AC	F0	RS1	RS0	OV		P

Bytes 3

Cycles 2

Encoding 10000101 src_direct dest_direct

Operation MOV
(dest_direct) = (src_direct)

Example MOV P1, P0

MOV direct, #immediate

C	AC	F0	RS1	RS0	OV		P

Bytes 3

Cycles 2

Encoding 01110101 direct immediate

Operation MOV
(direct) = immediate

Example MOV P2, #0FFh

MOV direct, @Rn

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 2

Encoding 1000011n direct

Operation MOV
(direct) = (Rn)

Example MOV P0, @R1

MOV direct, A

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 1

Encoding 11110101 direct

Operation
MOV
(direct) = A

Example MOV P0, A

MOV direct, Rn

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 2

Encoding 10001nnn direct

Operation
MOV
(direct) = Rn

Example MOV P2, R5

MOV DPTR, #immediate

C	AC	F0	RS1	RS0	OV		P

Bytes 3

Cycles 2

Encoding 10010000 $immediate_{15-8}$ $immediate_{7-0}$

Operation
MOV
DPTR = immediate

Example MOV DPTR, #1234h

MOV Rn, #immediate

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 1

Encoding 01111nnn immediate

Operation MOV
Rn = immediate

Example MOV R4, #0h

MOV Rn, A

C	AC	F0	RS1	RS0	OV		P

Bytes 1

Cycles 1

Encoding 11111nnn

Operation MOV
Rn = A

Example MOV R5, A

MOV Rn, direct

C	AC	F0	RS1	RS0	OV		P

Bytes 2

Cycles 2

Encoding 10101nnn direct

Operation MOV
Rn = (direct)

Example MOV R4, P1

JB

The JB instruction branches to the address specified in the second operand if the value of the bit specified in the first operand is 1. The bit that is tested is not modified. No flags are affected by this instruction.

JB bit, offset

C	AC	F0	RS1	RS0	OV		P

Bytes	3		
Cycles	2		
Encoding	00100000	bit	offset
Operation	JB PC = PC + 3 IF (bit) = 1 PC = PC + offset		
Example	JB P1.2 LABEL		

2 Этап. Структурная схема и общее описание её работы.
Построение структурной (блок) схемы – первый этап в проектировании схемы ЭВМ на основе программной модели.

Следующие принципы учтены при выборе структуры:

1. Иерархический подход к проектированию схем, который поддерживается в MaxPlus. На первом этапе выполняется функциональное неформальное разбиение схемы на функциональные блоки с учетом распределения памяти по блокам и функциональным элементам, определяемым в программной модели.

2. Используется шинная организация соединений, достоинствами которой являются:
    a. максимально параллельное исполнение разнообразных передач между регистрами, регистрами и блоками иерархической памяти и выполнение элементарных операций в АЛУ;
    b. Используется регулярная схема управления, в которой применяется адресация при выборе регистров и определении функций записи и чтения, вместо одиночных управляющих сигналов. При этом можно ожидать более простую схему кодирования и декодирования микрокоманд.

3. Применяются, по возможности, простые регистры-защелки для хранения выбранных из памяти данных и промежуточных результатов. Операции счета и сдвига могут быть выполнены комбинационными схемами при передаче данных между регистрами.

4. Для максимально параллельного выполнения операций счета и сдвига используются накапливающие синхронизированные регистры-счетчики и сдвигатели. Выполнение этих микроопераций совмещается с передачами между регистрами и памятью, в которых активно используются шины.

5. К регистрам может быть обеспечен как регулярный адресный доступ через мультиплексоры, так и непосредственный для контроля и работы с отдельными битами и полями битов.

6. Неявно используемые регистры SFR для сокращения обращения к памяти дублируются с использованием их теневого отображения в памяти и непосредственного доступа в схеме при чтении.

Таким образом, сначала выбирается блочная структура памяти с учетом ее организации в программной модели (микроархитектуре). В один блок объединяются элементы памяти с одинаковыми интерфейсами. Признаками интерфейса являются – способ доступа (адресный – задаваемый режимами адресации в командах, адресный – через мультиплексоры, прямое обращение к регистрам), форматы и типы данных (слова, байты, биты).

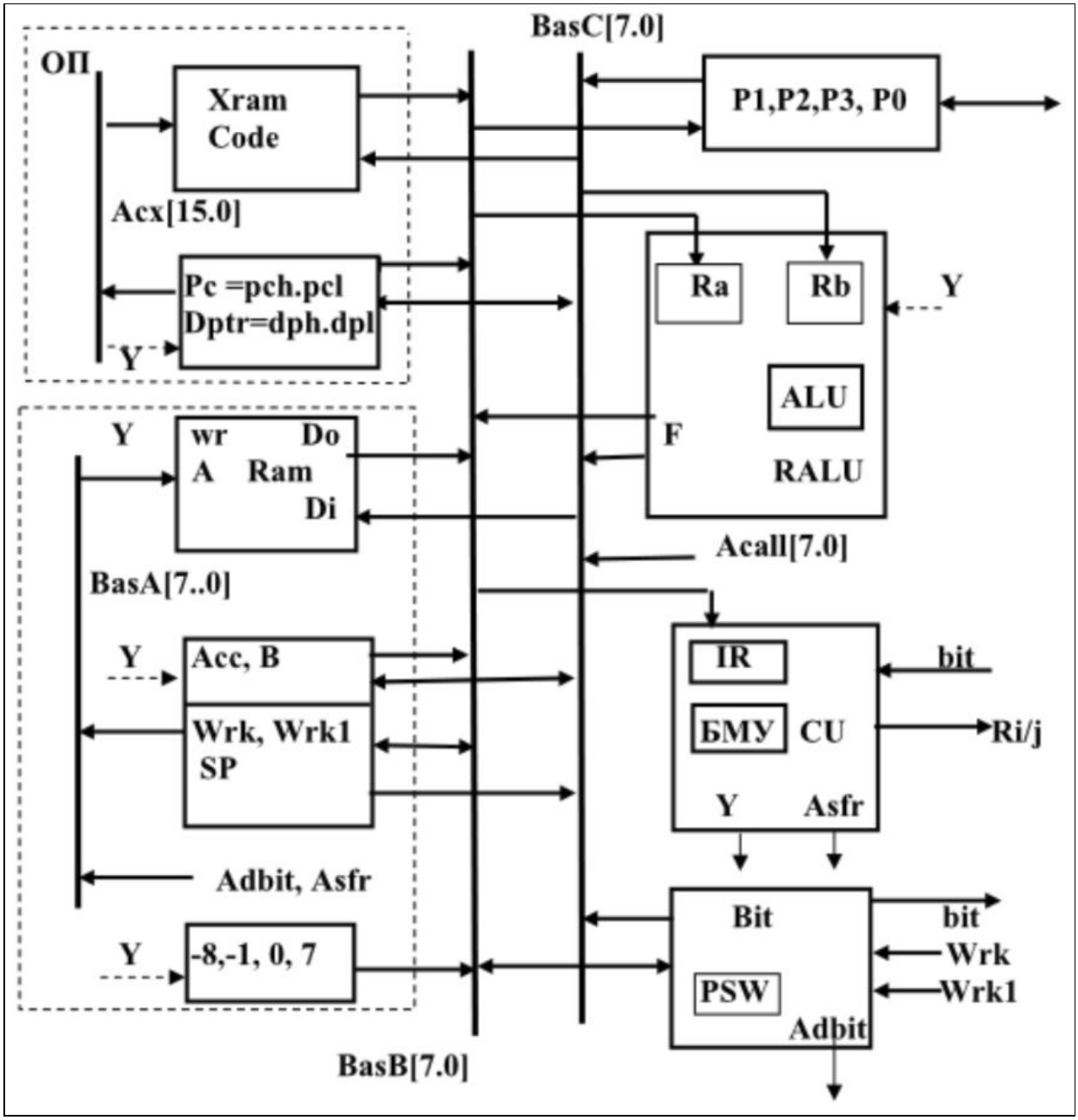


Схема работы

В схеме представлены функциональные устройства, специальные блоки для преобразования битовых данных и формирования констант.

- Шины данных и адреса = Acx[15.0], BasA[7.0], BasB[7.0], BasC[7.0].

Блок основной памяти ОП. Устройство управления CU. Регистровое арифметико-логическое устройство РАЛУ. Y – сигналы управления блоками и устройствами ЭВМ формируются в CU.

1. Блок основной памяти (ОП) включает:
    a. Постоянную программную память Code(ограничиваемся 256 байтами).
    b. Память данных Xdata (256 байт).
    c. В ОП используется общая адресная 16-битовая шина ACX[15.0], слово памяти – байт.
    d. 16-битовые адресные регистры-счетчики с прямым и адресным доступом (DPTR, PC), которые формируют 16-битовый адрес на адресной шине ACX[15.0].

2. Блок внутренней быстрой памяти:
    a. RAM(256 байт) объединяет Data и SFR с общей 8-битовой шиной адреса BasA[7..0] и 8-разрядным словом данных.
    b. 8-битовые арифметические регистры с прямым и адресным доступом (ACC, B), используемые в арифметических и логических операциях, регистры имеют теневое отображение в SFR.
    c. Адресные и рабочие регистры (SP, Wrk, Wrk1) – формируют 8-битовый адрес, хранят операнды и параметры команды, являются счетчиками циклов в операциях умножения и деления и сдвигателями. SP имеет теневое отображение в SFR.

3. Регистровое арифметико-логическое устройство (RALU) включает:
    a. Арифметико-логическое устройство (АЛУ)
    b. Регистры временного хранения операндов RA, RB.

4. Устройство управления (CU) содержит:
    a. Регистр команд IR
    b. Блок микропрограммного управления (БМУ) с декодером микрокоманд Y.

5. Блок двунаправленных портов ввода-вывода (P0,P1,P2,P3) связан с внешними контактами микросхемы, содержит одноименные регистры с прямым доступом и с теневым отображением в SFR.

6. Блок формирования констант (0, -1, -8, 7).
7. Блок выборки и выполнения битовых операций BIT. Блок подключается к рабочим регистрам Wrk,Wrk1 и содержит регистр PSW. В блоке формируется адрес доступа к битам в Adbit, значение бита BIT для условных микрокоманд в CU.

Для соединения модулей памяти, регистров и других функциональных элементов используются шины, мультиплексоры и селекторы.

Мультиплексирование – физическое подключение элементов (в пространстве) к общей шине, включая последовательный во времени адресный выбор и подключение элементов к шине и запись с шины.

Каждый мультиплексор входных данных BasB, BasC, BasA, ACX позволяет прочитать по адресу данные только из одного источника (регистра, памяти).
Запись с мультиплексированных шин в регистры и память выбирается адресным декодером (селектором) WrB – с шины BasB, декодером WrC – с шины BasC. Сигнал записи обозначается единицей на одном из 2n выходов декодера, где n-разрядность адреса выбираемого функционального элемента на шинах BasB и/или BasC.
При этом нуль на всех остальных выходах декодера обозначает параллельное чтение, но выбор одного из читаемых значений осуществляется мультиплексором шины.

Этап 3. Код программы

```
#include <reg51.h>
#define MEM_COUNT 128

typedef struct {
  int value;
  int capacity;
} MemoryCell, Register;

MemoryCell xdata Memory[MEM_COUNT];

Register xdata Akk, CommandReg;

void orl(Register* a, Register* b, char is_ref){
  int val_b = 0;

  if (is_ref) {
    val_b = Memory[b->value].value;
  } else {
    val_b = b->value;
  }

  a->value = a->value | val_b;
}

void jb(int b, int addr){
  if ((Akk.value >> b) & 1) {
    CommandReg.value = addr;
  }
}

void mov(Register* a, void* b, char is_b_reg, char is_a_ref, char is_b_ref){
  int val_b=0;
  if (is_b_reg) {
    val_b = ((Register*)b)->value;
  } else {
    val_b = (int)b;
  }

  if (is_b_ref) {
    val_b = Memory[val_b].value;
  }

  if(is_a_ref) {
    Memory[a->value].value = val_b;
  } else {
    a->value = val_b;
  }
}
```

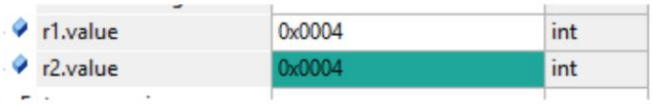
```
int help_dec(int val, int cap){
  if (val == 0) {
    return (2<< (cap))-1;
  }

  return val-1;
}

void dec(Register* reg, char is_ref){
  if (is_ref){
    Memory[reg->value].value = help_dec(Memory[reg->value].value,
Memory[reg->value].capacity);
  return;
  }

  reg->value = help_dec(reg->value, reg->capacity);
}

main() {
  // Initialization
  int i = 0;
  Register r1,r2;
  Akk.capacity = 16;
  CommandReg.capacity = 16;
  for (i=0;i<MEM_COUNT;i++) {
    Memory[i].capacity = 16;
  }

  r1.capacity = 16;
  r2.capacity = 16;

  // Tests
  dec(&r1,0);
  mov(&r1,4,0,0,0); // r1.value = 3
  mov(&r2, &r1, 1, 0, 0); // r2.value = r1.value
  dec(&r2, 0); // r2.value --;
  orl(&r1, &r2, 0); // r1.value |= r2.value
  mov(&r1, 23, 0, 1, 0); // mem[r1->val] = 23
  Akk.value = 7;
  jb(2, 25);

  return 0;
}
```

Этапы 4. Выполнение

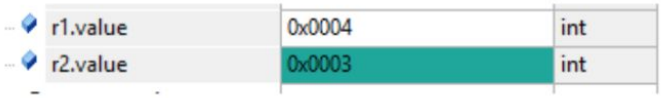
dec(&r1,0); (r1 = 0)

mov(&r1,4,0,0,0);

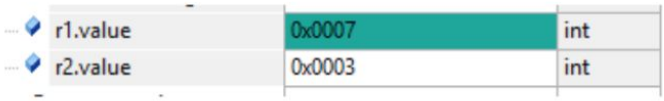
mov(&r2, &r1, 1, 0, 0);



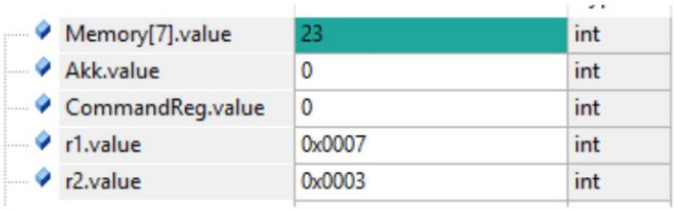
dec(&r2, 0); (r2 = 4)



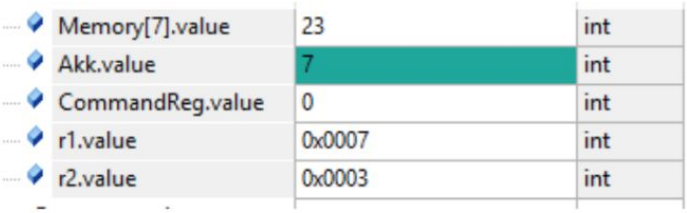
orl(&r1, &r2, 0);



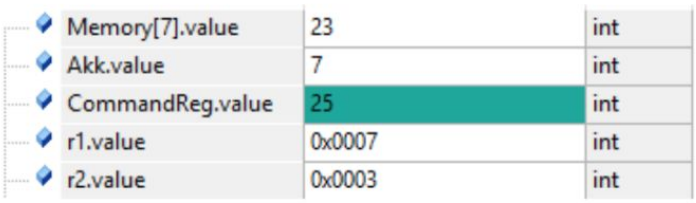
mov(&r1, 23, 0, 1, 0);



Akk.value = 7;



jb(2, 25);



Вывод:

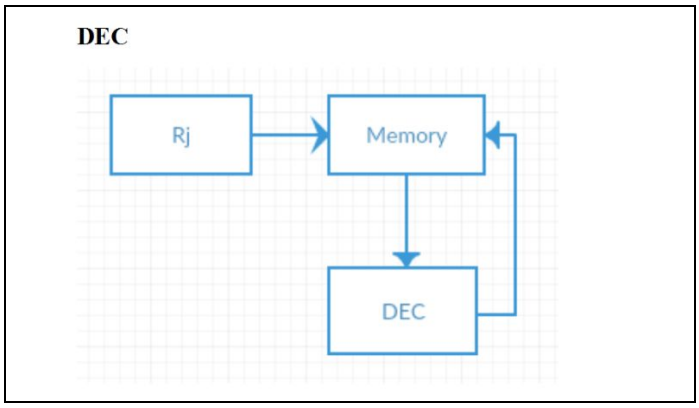
Теория подтвердилась практикой.

Были реализованы и протестированы следующие компоненты: mov, dec, jb и orl.

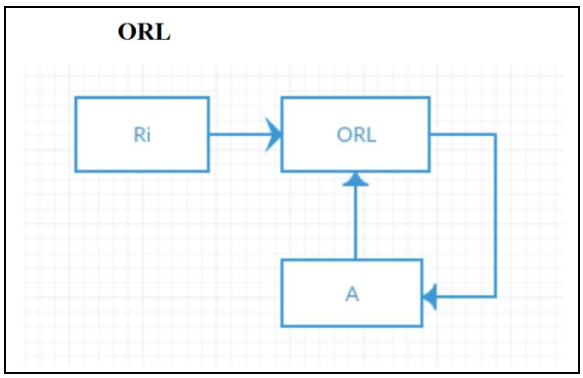
Этапы 5 и 6. Построение схем.

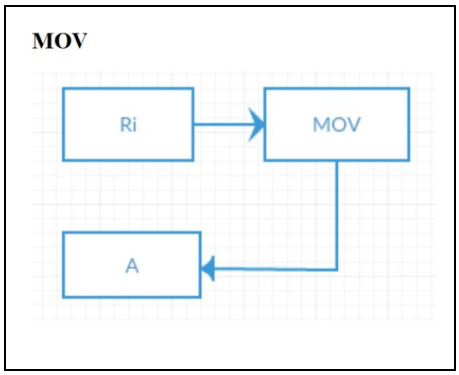
Цель:
построить цифровые схемы, реализующие заданный набор команд. Для пятого этапа необходимо построить схему реализации системы с одной инструкцией. Для шестого этапа необходимо построение схемы системы, реализующий полный набор команд.

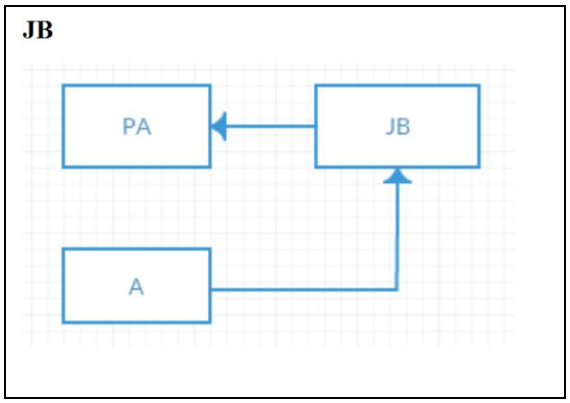
Построение схемы для одной команды:



Построение схемы оставшихся команд:







Этап 7

Цель:
В данном этапе необходимо дать оценку кол-ву элементов, требуемых для реализации системы, выведенной в предыдущем этапе.

Для реализации схем выполнения команд необходимы следующие элементы:

- РК - 8 бит
- РА - 8 бит
- РФ - 8 бит
- Аккумулятор - 8 бит
- Счетчик команд 8 бит
- Два мультиплексора 8:1 для адресации битов аккумулятора и регистра флагов
- Мультиплексор 256:1 для выборки ячейки памяти
- АЛУ Регистр
- 8 элементов ИЛИ
- Декремент (2 * XOR + НЕ)

Для каждого бита регистра нужен один D-триггер.
Каждый D-триггер состоит из четырех логических элементов И.
Мультиплексор 8:1 состоит из 8-ми элементов И, 3-х элементов НЕ и одного элемента ИЛИ (12 элементов).
Для ORL необходимы 8 элементов ИЛИ, для декремента 8 стандартных элементов и НЕ.

Количество необходимых базовых логических элементов будет равно: N = 452

Заключение
В ходе выполнения курсовой работы были освоены основные навыки проектирования, изучены возможные наборы инструкций, особенности их работы, реализации, и документирования. Кроме того, были закреплены умения работы в среде Keil uVision.