Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
Томский государственный университет систем управления и радиоэлектроники

Кафедра комплексной информационной безопасности
электронно-вычислительных средств

П.Н. Коваленко

Методические указания по выполнению лабораторных работ

по дисциплинам «Аппаратные средства вычислительной техники»,
«Проектирование центральных и периферийных устройств электронно-вычислительных систем»,
«Организация электронно-вычислительных машин и систем»

В-Спектр
Томск – 2012

УДК 004
К 56

К 56 **Коваленко П.Н.** Методические указания по выполнению лабораторных работ. – Томск.: В-Спектр, 2012. – 80 с.
ISBN 978-5-91191-245-7

Методические указания содержат изложение требований и примеров выполнения разделов пояснительной записке при выполнении лабораторных работ по дисциплинам «Аппаратные средства вычислительной техники», «Проектирование центральных и периферийных устройств электронно-вычислительных систем», «Организация электронно-вычислительных машин и систем».

Предназначено для студентов направления 090100 – «Информационная безопасность», 210200 – «Проектирование и технология электронных средств», 230100 – «Информатика и вычислительная техника».

УДК 004

ISBN 978-5-91191-245-7

ОГЛАВЛЕНИЕ

Лабораторная работа № 1

Цель работы:
Изучить принцип последовательного программного управления на основе игровой модели управления роботом.

1. Введение

Принципы построения ЭВМ, придуманные и опубликованные в научной статье группой ученых в составе с Джоном Фон Нейманом дали новый толчок в развитии вычислительной техники 40х годов 20го века. Сегодня трудно найти такую вычислительную машину, в которой не использовались бы эти принципы. Например, предложенный принцип двоичного кодирования (данные в ЭВМ представляются в бинарной форме, в виде «0» и «1») широко используется не только в каждом микропроцессоре, но и в цифровой электронике вообще.

Другие принципы конкретизируют строение и поведение вычислительной машины, как например, принцип однородности памяти, который описывает строение памяти машины (программы и данные находятся в одной памяти) и в настоящее время не применяется, как возможно и принцип жесткости архитектуры в будущем.

Принцип последовательного программного управления раскрывает поведение ЭВМ в процессе его работы: все команды располагаются в памяти и выполняются последовательно, одна после другой, в последовательности, определяемой программой.

Структура ЭВМ состоит из основных элементов: памяти, в которой располагаются команды, дешифратора, который определяет тип действия из кода команды и исполнительного устройства, который выполняет действие. На рисунке 1 представлена условная структурная схема ЭВМ и фазы ее работы.

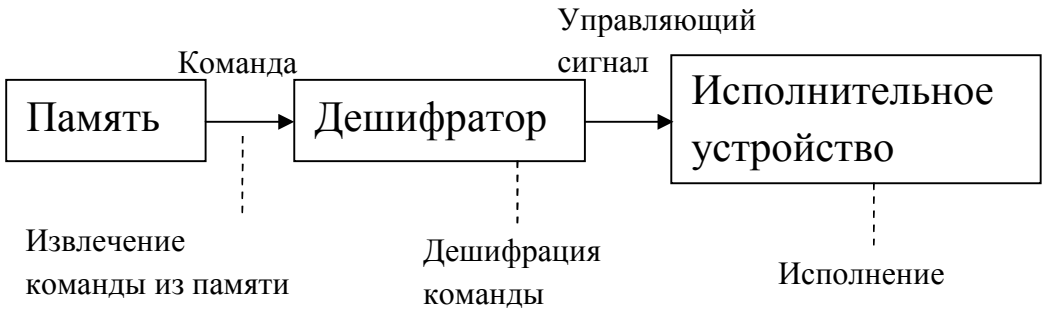


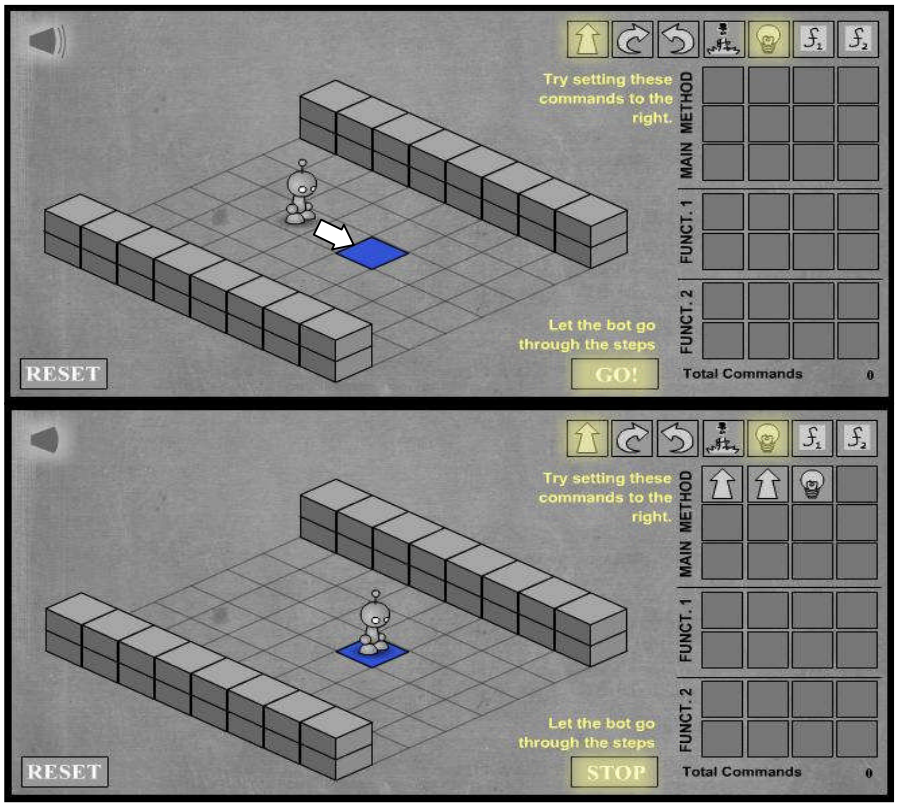
Рисунок 1 – Условная структура ЭВМ и фазы ее работы

Понять, как работают машины, построенные по этому принципу, позволяет данная лабораторная работа.

2. Описание игровой модели управления роботом

В основу игровой модели положено поле, которое разбито на квадраты и изображено в изометрической проекции, как показано на рисунке 2. На поле размещен робот, которым управляет пользователь. Игровая модель состоит из 12 задач, сложность которых возрастает по мере их выполнения.

Для того, чтобы выполнить задачу, необходимо зажечь все синие квадраты на поле. Для этого, нужно подвести робота на квадрат и зажечь лампочку.



а)

б)

Рисунок 2 – Внешний вид игровой модели,
где: а) начальное положение, б) конечное положение робота

Управление роботом осуществляется при помощи программы, которая состоит из множества последовательно – исполняемых команд. В правой части экрана показана область управления – память, в которую пользователь записывает управляющие команды. Это осуществляется при помощи перетаскивания «мышью» иконок из правой верхней части экрана в соответствующую область памяти.

Для того, чтобы проверить правильность работы алгоритма программы, пользователь может нажать кнопку «GO!», тем самым, запустив программу на исполнение. Останов программы

осуществляется кнопкой «STOP». Очистка памяти робота производится при помощи кнопки «RESET».

3. Система команд

Команды условно разделяются на два типа: исполнительные и сервисные. Исполнительные команды предназначены для перемещения положения робота и воздействия на окружающую среду (включить/выключить лампу). Сервисные команды предназначены для организации программных функций и являются указателями на функциональную область памяти. При выполнении такой команды управление перейдет к тому участку программы, которая записана в соответствующей области памяти. Ниже дано описание каждой команды.

Исполнительные команды:

 Робот двигается вперед на одну клетку

 Робот поворачивается по часовой стрелке на 90^0

 Робот поворачивается против часовой стрелки на 90^0

 Робот запрыгивает на ступень или спрыгивает с нее

 Находясь на синем квадрате, засвечивает его, либо тушит, если квадрат уже был засвечен.

Сервисные команды:

 Указатель на память функции 1

 Указатель на память функции 2

4. Организация памяти

Команды управления роботом могут располагаться в любой области его памяти. Память условно разделена на две области: основную и функциональную. Робот выполняет только те команды, которые расположены в основной памяти. Сервисные команды, расположенные в основной памяти позволяют исполнять команды из памяти функций.

Т.е. если вызвать команду f1, то будут выполняться команды из памяти функций 1, если команда f2, то из памяти функций 2 соответственно.

На рисунке 3 показана организация памяти робота. Команды функций можно вызвать и в памяти другой функции (не имеет смысла вызывать их в той же области памяти, т.к. организуется цикл, не имеющий выхода).

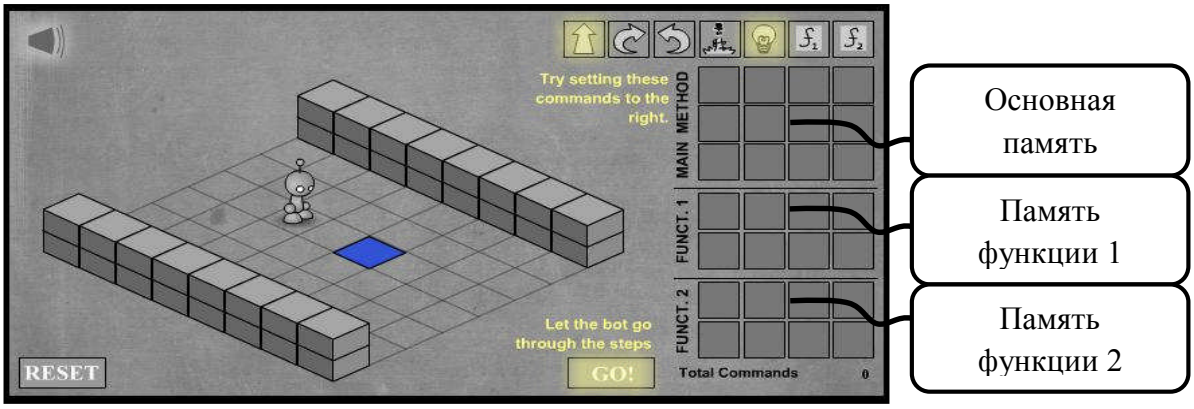
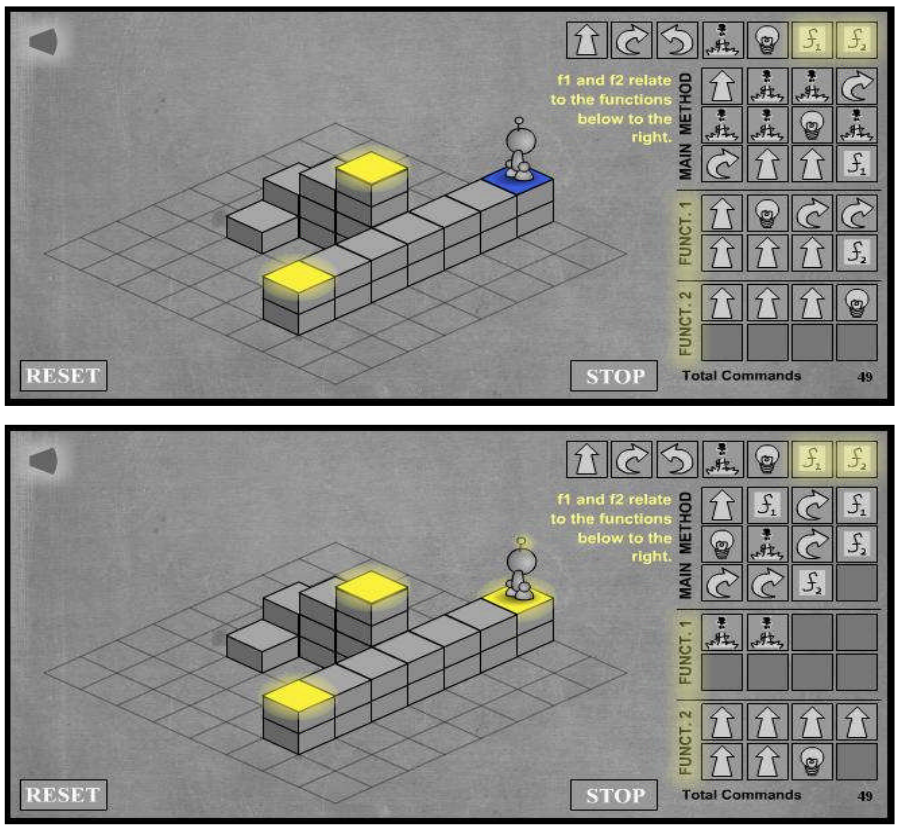


Рисунок 3 – Организация памяти робота

Для выполнения задачи надо определить оптимальный путь робота и составить программу.

На рисунке 4 приведены примеры прохождения одной и тоже задачи. Можно заметить, насколько эффективно используется память. В первом случае, (рисунок 4а) память функций используется как дополнение к основной памяти и повторное использование функций не предусматривается. Во втором случае, используется функциональный подход в программировании. Такой подход позволяет эффективно использовать память программ за счет повторного вызова функций.



а)

б)

Рисунок 4 – Примеры выполнения задачи при различном использовании памяти функций,
где: а – однократные вызовы функций, б – многократные

Решение следующих задач невозможно без вторичного использования функций. Эта ситуация возникает потому что количество ячеек памяти ограничено. На рисунке 5 показана такая ситуация, когда памяти недостаточно для прохождения уровня «в лоб».

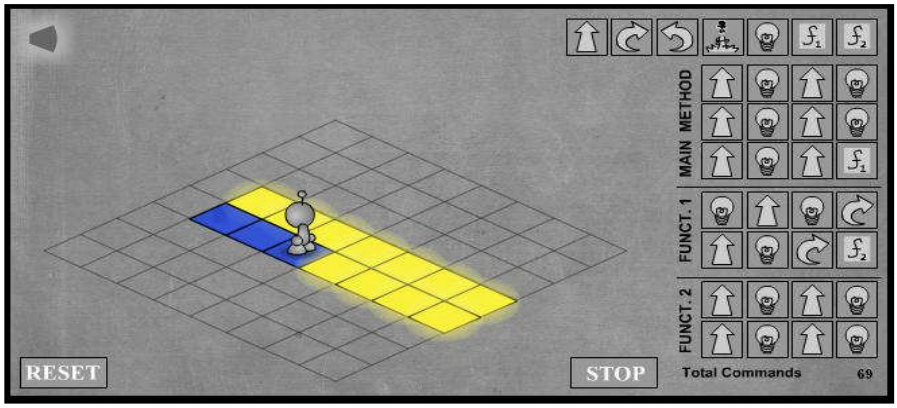


Рисунок 5 – Вариант неверного решения задачи, при котором возникает дефицит памяти

Для решения этой задачи и следующих, необходимо составить оптимальный маршрут движения и определить повторяющиеся участки программы, которые можно записать в память функций и использовать их повторно.

5. Задание к лабораторной работе

- Ознакомиться с игровой моделью управления роботом. (Организация памяти, команды управления, интерфейс пользователя);
- Решить задачу 11-го уровня;
- Предложить состав и алгоритм работы устройства управления роботом;
- Подготовить отчет по лабораторной работе;

6. Содержание отчета

В отчете необходимо привести следующие сведения:

- Цель лабораторной работы;
- Описание игровой модели управления роботом (интерфейс пользователя, органы управления, описание команд);
- Предложенный алгоритм и структуру устройства управления роботом;
- Выводы.

7. Вопросы для самоконтроля

- Какие функциональные команды робота управляют его перемещением?
- В чем заключается принцип последовательного программного управления?
- Какие фазы в работе устройства управления роботом можно выделить?
- Зачем нужны команды f1 и f2?
- Как нужно оптимизировать программу для того, чтобы решить задачи последних уровней?
- В каких случаях можно использовать вторичное использование функций?

8. Список литературы

1. Русанов В. В. Микропроцессорные устройства и системы (МПУиС) : учебное пособие для студентов направления 210100 "Электроника и микроэлектроника" специальности 210106 "Промышленная электроника" / В. В. Русанов, М. Ю. Шевелев ; Федеральное агентство по образованию, Томский государственный университет систем управления и

радиоэлектроники, Кафедра промышленной электроники. - Томск : ТУСУР, 2012. - 184[2] с. : ил. - Библиогр.: с. 184. Электронный ресурс –[www.edu.tusur.ru/training/publications/867]
2. Puzzle game «light-bot»[Электронный ресурс] www.armorgames.com/play/2205/light-bot

Лабораторная работа № 2

Содержание

Цель работы – Изучить структурную и функциональную организацию машины Поста.

1. Введение.

Эрнст Пост представлял, что данные, обрабатываемые машиной, размещены на ленте «бесконечной» длины, поделенной на одинаковые секции. Такое представление данных естественно, поскольку свою гипотезу он выдвинул в эпоху бурного развития телеграфной связи (ввод-вывод данных осуществлялся на перфорированную ленту).

Машина Поста состоит из неподвижной ленты и каретки. По ленте влево - вправо движется сенсорная, чувствительная каретка с возможностью записи данных («1» или «0») в секции и их чтения. Каретка в неподвижном состоянии находится на одной секции, а за единицу времени (такт), по команде, каретка может сместиться только на одну секцию. Состояние ленты может меняться в процессе работы машины. Тогда, состояние машины – это состояние ленты и положение каретки (номер секции на которой находится каретка).

Различают начальное и конечное состояния машины. Эти состояния определяются условием прикладной задачи. Рассматривая задачи, решаемые с применением машины Поста, будем говорить о начальном состоянии ленты и положении каретки, а после действия - о конечном состоянии ленты и положении каретки. Например, начальное состояние ленты не изменилось, но каретка переместилась на один шаг (позицию) и изменилось значение в секции ленты, в которую находится каретка.

Требование к переводу машины из начального в конечное состояние определяет цель действия. Эта цель формулируется каждый раз, как возникает новая прикладная задача.

2. Структурная организация элементов машины Поста.

Структурная схема модели машины Поста представлена на рисунке 1.

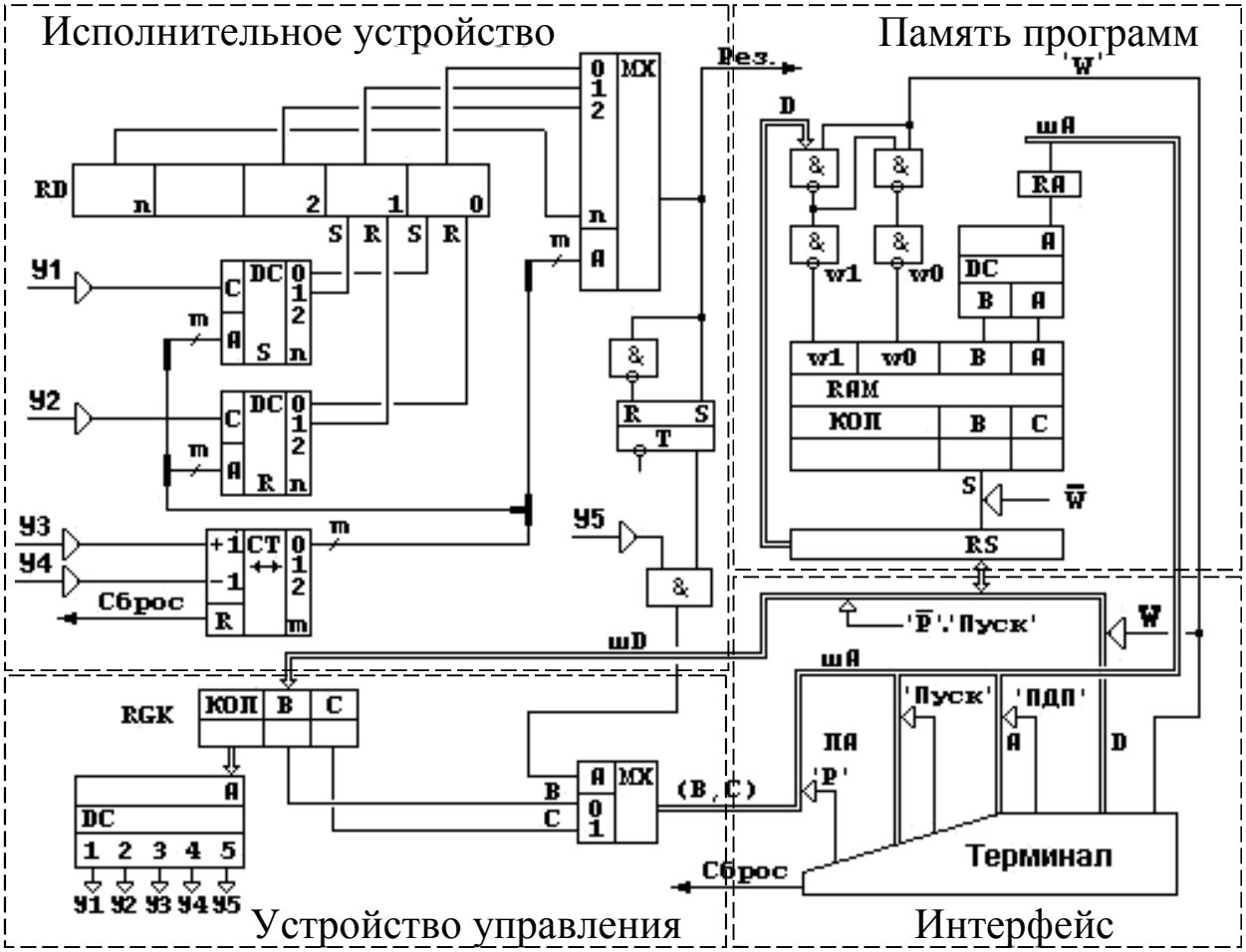


Рисунок 1 – Структурная схема модели машины Поста

Как видно из рисунка, модель машины Поста состоит из основных блоков:

- Интерфейса, который предназначен для организации пользователя с машиной;
- Памяти программ, которая предназначена для хранения команд пользователя;
- Управляющего устройства, которое производит дешифрацию команды и создает управляющие сигналы для их выполнения;
- Исполнительного устройства, которое исполняет команду пользователя, производя действия исходя из управляющих сигналов.

Для того, чтобы имитировать ленту машины Поста достаточно иметь регистр данных RD, который можно построить при помощи RS-триггеров. В этом случае одной секции соответствует один триггер, который из которых может хранить бинарную информацию («1» или «0»).

Каретку можно реализовать при помощи двух дешифраторов DC и мультиплексора MX, которые соединены с регистром данных.

Перемещение каретки может задаваться при помощи адреса, генерируемого счетчиком СТ.

Синхронизация операции записи/чтения и перемещения каретки определяется программой или последовательностью команд машины. Действия каретки определены типом операции, указанной в команде.

Устройство управления определяет тип операции, хранимой в регистре команд (RGK), и вырабатывает с помощью дешифратора команд DC(1-5) соответствующие синхронизирующие сигналы Y(1-5).

Запись программы осуществляется при помощи терминала с возможностью непосредственного обращения к памяти программ. Выборка команд из этой памяти осуществляется благодаря адресации к памяти программ и порта ввода-вывода. Так, например, на начальном этапе работы машины, после того, как был произведен аппаратный сброс, пусковой адрес устанавливается равным 1 и далее он определяется при помощи адресной части команды (B или C операнд).

3. Машина Поста и ЭВМ

3.1. Сходство с ЭВМ

- Вся информация, обрабатываемая в машине, представляется в двоичном виде и распределяется в элементах памяти.
- Как для машины Поста, так и для ЭВМ, указывается некоторый ограниченный набор элементарных операций (действий). За один шаг, ЭВМ, как и машина Поста, может совершить какое-либо действие из этого набора.
- Машина Поста, как и ЭВМ, работает по программе, указывающей, какие действия, и в каком порядке должны быть выполнены.

3.2. Отличия от ЭВМ

- Доступ к данным в машине Поста возможен только линейно (последовательный доступ), тогда как ЭВМ имеет ОЗУ с произвольным доступом. Чтобы из одной секции перейти к другой, каретка должна пройти все промежуточные секции.
- В архитектуре ЭВМ, при выполнении программы, порядок выполнения команд определяется исходя из внутреннего состояния специального регистра – счетчика команд, тогда как в машине Поста

последовательность выполнения команд определяется в самой программе (В операнд).

4. Организация машины Поста

4.1. Исполнительное устройство

Исполнительное устройство (ИУ) включает в себя имитатор ленты и имитатор каретки. Имитатор ленты представляет собой набор триггеров, каждый из которых может хранить бит информации. Лента является общим понятием хранилища данных. Современные вычислительные средства реализуют функцию хранения посредством регистров.

Регистр данных (RD), имитирует секции ленты, представлен в виде набора триггеров, каждый из которых имеет два входа: первый (S) - запись «1», второй (R) -запись «0». Прямой выход триггера отображает состояние триггера, т.е. после того, как была произведена запись значения в триггер, это значение будет представлено на его выходе.

Имитатор каретки обеспечивает позиционирование напротив активной секции. Если пронумеровать секции, то каретка должна последовательно обращаться к заданным номерам (например, начальное положение каретки).

Адресация активной секции является функцией счетчика секций (СчС). Так как счетчик секций осуществляет двоичный счет, то на базе счетчика можно имитировать сдвиги каретки влево (СчС: = СчС+1) или вправо (СчС: = СчС-1).

Операции записи обеспечиваются передачей управляющих сигналов на соответствующий вход R или S триггера, что позволяет записать в активную секцию «0» или «1». Чтение состояния секции обеспечивается коммутатором (мультиплексор), для которого адрес активной секции указывает счетчик секций.

4.2. Устройство управления

Устройство управления (УУ) в соответствии с его функциями хранит слово «команда», пока не закончено ее исполнение. Поэтому для хранения информации команды можно использовать специальный регистр команд (RGK). В RGK выделим три поля: поле КОП –

действия, поле В - нижней и С - верхней отсылки к следующим командам.

Имитатор УУ содержит коммутатор отсылок В и С, которые указывают на адрес следующей команды. Выбор отсылки определен состоянием линии управления (ЛУ), которое вычисляется ИУ при выполнении команды в зависимости от состояния активной секции ленты и сигнала У5 по логике «И».

Отсылка, выбранная с помощью коммутатора, В - нижняя или С - верхняя, является адресом, который поступает в память для выборки очередной команды. Команды, сформированные списком, как показаны в таблице 1. Порядковый номер в списке определяет код операции (КОП). В столбце КОП показана двоичная запись одноименного номера.

В таблице 1 выделены следующие группы операций:

позиции 1-2 - группа «Запись» в активную ячейку;

позиции 3-4 - группа «Сдвиг»;

позиция 5 - команда «Решение»;

позиция 6 - команда «Останов».

Таблица 1

п/п	Оператор (КОП)		Сигналы микроопераций				
	Мнемоника КОП	Двоичный код КОП	Y1	Y2	Y3	Y4	Y5
1	Запись «1»	001	1	0	0	0	0
2	Запись «0»	010	0	1	0	0	0
3	Сдвиг ←	011	0	0	1	0	0
4	Сдвиг →	100	0	0	0	1	0
5	1/C ⟨RD = 1⟩? 0/B	101	0	0	0	0	1
6	Останов	000	0	0	0	0	0

При выполнении команды «Останов» никакие управляющие сигналы не генерируются и выполнение программы прекращается.

В графе «Сигналы микроопераций» указаны наименования сигналов управления и момент их активизации - логическая «1».

Эти сигналы могут быть отображены функцией, которая реализуется дешифратором команд DC, как показано на рисунке 4.

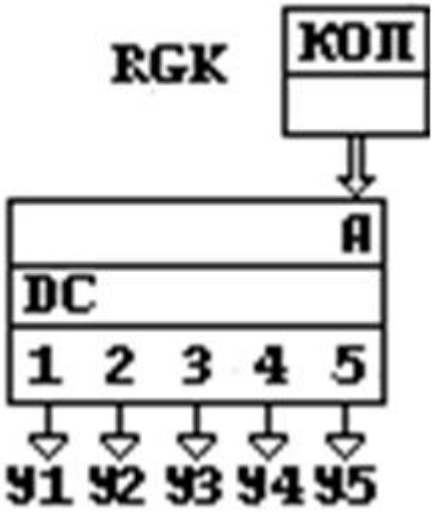


Рисунок 2 – Дешифратор команд

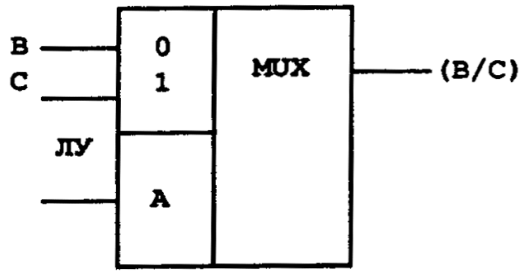


Рисунок 2 - Мультиплексор отсылок

Таблица 2

A	MUX
0 1	B C

В состав УУ входит коммутатор отсылок, который выполнен при помощи мультиплексора, как показано на рисунке 2. Его таблица истинности приведена в таблице 2. Графа «А» таблицы 2 назначается после вычисления логических условий (ЛУ) в операционном автомате, где выделяется состояние активной секции («0», «1»). Это состояние передается в графу «А» при выполнении команды «Решение». Однако, при выполнении других команд, значение ЛУ определено нулю. Графа MUX определяет значение выхода функции. Это значение соответствует отсылкам B или C, коммутируемым в зависимости от значения графы А.

4.3. Терминал

Терминал предназначен для указания режима работы (ПДП или ВЫЧИСЛЕНИЯ), а также для управления «Пуском» машины или продолжением выполнении программы. Кроме того, с пульта управления оператор указывает пусковой адрес (ПА).

4.4. Память программ

Память программ предназначена для хранения команд пользователя. Запись этих команд осуществляется при помощи терминала в режиме прямого доступа к памяти. При выполнении программы, по шине данных передается адрес команды, по которому из

определенной ячейки памяти извлекается команда, которая отображается на шине данных.

5. Структура машины Поста

При детальном рассмотрении рисунка 1 «слева направо» и «сверху вниз, можно заметить, что регистр данных RD(n-0) в каждом разряде (секции) имеет R-S триггер, входы S триггеров подключены к соответствующим выходам дешифратора DCS, а входы R – к соответствующим выходам дешифратора DCR. Например, сигналы DCS(0) и DCR(0) подключены соответственно ко входам S и R нулевого разряда регистра RD(n-0). Выходы RD(n-0) подключены ко входам мультиплексора MX, выход которого образует результат «РЕЗ» и подключен к R-S-триггеру, а выход этого R-S-триггера связан с одним входом логической схемы «И». Состояние выхода элемента «И» определяет условия выбора номера следующей команды программы. Выход элемента «И» подключен к адресной зоне мультиплексора MX. (т.н. мультиплексор отсылок). Из рисунка 1 видно, что такие отсылки названы B и C (выход мультиплексора). Слова B и C поступают с выхода регистра команд RGK.

К адресным зонам дешифраторов DC и мультиплексора MX подключен выход счетчика CT. Состояние этого счетчика указывает адрес секции RD(n-0), с которой работает процессор (активная секция в смысле Поста). Можно заметить, что адрес может изменяться лишь на одно значение (+1, -1). Вход СБРОС этого счетчика CT, управляется с пультового терминала. Это способ установки начального состояния машины Поста, в частности каретки.

Дешифраторы DCS и DCR использованы для записи «1» и «0» в секцию регистра, а мультиплексор для чтения состояния секции, указанной счетчиком CT. Входы C (синхронизация) дешифраторов и входы (+1, -1) счетчика подключены к источнику управляющих сигналов (У), хотя на рисунке это явно не показано. Такие сигналы вырабатывает дешифратор кодов операций (КОП), подключенный к полю КОП регистра команд RGK. Сигнал У5 поступает на другой вход логического элемента «И», который определяет условие выбора отсылки B или C.

Справа на рисунке 1 дано обобщенное описание оперативного запоминающего устройства (RAM) и терминала. Оперативное запоминающее устройство связано посредством шины адресной (шA) и шины данных (шD) с процессором (все, что размещено на рисунке левее ОЗУ и терминала).

Шина адреса (шA) подключена ко входу регистру адреса (RA) памяти ОЗУ и соединена с выходом мультиплексора отсылок и пультовым терминалом. Шину шA загружают отсылкой (B, C) с выхода мультиплексора отсылок по команде «P» (продолжить), которая формируется с пультового терминала.

Шина шD связывает ОЗУ с регистром RGK команд и терминалом. ОЗУ имеет порт ввода-вывода RS (регистр слова). Передача слова из порта RS на RGK возможна при условии, что клавиша P (продолжить) не нажата, другими словами, команда «P» пультового терминала не введена, и был произведен запуск программы («ПУСК»). Через порт RS в память по шине шD данные вводятся с пультового терминала (управляющий сигнал «W»).

Ко входам WO и W1 элемента RAM подключена схема (четыре логических элемента) выбора режима «чтение-запись» работы ОЗУ. Эта схема синхронизируется сигналом «W», т.е. когда W=1 - запись, иначе - чтение. К выходу регистра RA адреса ОЗУ подключен дешифратор, имеющий два выхода A и B, которые указывают адрес запоминающего элемента, установленного на пересечении столбцов B и строк A матрицы RAM.

6. Модель «Машина Поста»

Модель разработана с целью демонстрации одной из традиционных формализаций понятия процессор наряду с такими формализациями, как машина Тьюринга и т.п.

Управление моделирующей программой осуществляется в диалоговом режиме экранного редактирования путем выбора соответствующего пункта меню. Пояснительные надписи, комментирующие смысл необходимых действий, выводятся на экран.

Результаты работы программ представлены на экране в виде динамических картинок состояния элементов, узлов и устройств машины в процессе интерпретации команд программы пользователя.

При создании программной модели машины Поста в структуру обучающей системы были введены следующие ограничения:

- состав машины Поста определен минимальной конфигурацией, которая включает в себя: процессор, детализированный до уровня: триггер, регистры, мультиплексор, счетчик, шины; оперативное запоминающее устройство, детализированное до уровня: элемент памяти, матрица запоминающих элементов, адресные дешифраторы столбцов и строк матрицы, порт ввода-вывода, регистры и шины; пультовый терминал; системная магистраль, детализированная до уровня шина адреса, шина данных, сигнал управления записи/чтения;
- система команд (в смысле Поста) должна быть минимальной (не более шести), но достаточной для построения алгоритмических структур следования, ветвления и циклов;
- адресное пространство программной памяти - 99 десятичных слов (в модели ограничено 32 адресами, что достаточно для учебных целей), а регистр данных, т.е. лента в смысле Поста – 32-разрядный;
- формат команды содержит поле кода операции и поле адреса следующей команды, причем это поле представляет совокупность двух полей (в смысле Поста - верхняя и нижняя отсылки для команды «Решение»);
- режимов работы моделирующей программы - 2. Первый режим - ручной ввод программы пользователя и ввод исходных данных (соответствующий пункт «Меню») с возможностью сохранения введенной программы в памяти моделирующей ЭВМ и вызова ее в оперативную память, а также с возможностью редактирования программы и данных. Второй режим – исполнение программы в пошаговом или автоматическом режиме. Команды программы разделены символом 'P' - продолжение.

На первом этапе работы с обучающей системой предусмотрено изучение разделов: анализ системы команд (6 команд); организация ветвлений и циклов; примеры программирования.

На втором этапе, многоуровневая система меню предлагает исследователю:

- получить справочную информацию по работе и организации машины Поста;

- ознакомиться с примерами решения типовых задач (например, тест системы команд) и, при желании, повторить их на демонстрационных динамических рисунках;

При этом, система подсказок позволяет закрепить последовательность и содержание шагов решения задачи, а динамические рисунки отражают ситуации на объекте и создает эффект работы с реальной средой.

7. Запуск и работа с автоматизированной обучающей системой

АОС «Машина Поста» имеет удобный интерфейс. В директории АОС 0 найдите файл с именем «aos0.bat» и запустите его. Ознакомьтесь со структурной схемой машины, выделите основные блоки. Обратите внимание на функциональное назначение клавиш, показанных по периметру экрана. Начните с записи команды в память (клавиша F2) в соответствии с примером. Программа АОС 0 покажет Вам подробности процесса. При выполнении команд программы в режиме, определяемом клавишей F4, следите за подсказками.

Изучите структурную схему машины, выведенной на экран. Выделите основные элементы: УУ, ИУ, ОЗУ, терминал, системные шины. Обратите внимание на «окна» и функциональные клавиши.

Воспользуйтесь клавишей F6 «Пример работы». В меню выберите «Тест команд». В окне «Программа» появится текст программы. Проверьте ход исполнения шагов программы и соответствие со схемой алгоритма «TEST». Клавиша F4 «Запуск» откроет меню – «По шагам», ENTER. Следите за подсказками системы. Продвижение по шагам программы – клавиша «ПРОБЕЛ».

Программа АОС 0 позволяет решать три задачи:

- Анализ структуры машины: операционный и управляющий автоматы процессора; оперативная память; пультовый терминал; шины связи устройств.
- Организация шин адресов, данных и управления; структура оперативной памяти и организация записи/чтения данных через порт ввода-вывода.
- Изучение алгоритма работы машины Поста при интерпретации команд на примере TEST-программы системы команд (находится в

директории AOS0). Анализ алгоритма программы тестирования.

Синтез программы по заданному алгоритму модели элемента 2 И-НЕ.

Исследование поведения работы машины Поста следует с изучения схемы алгоритма, который представлен в виде блок-схемы на рисунке 3. Видно, что функционирование машины начинается с обращения к ОЗУ за выборкой команды программы (3). Далее осуществляется пересылка команды в процессор (4). Затем организована дешифрация поля формата команды (5, 6, 7, 8, 14, 15) и выполнение соответствующей микрооперации (9, 10, 11, 12, 16). При этом формируются логические условия (13, 17), по которым процессор определяет адрес следующей команды программы. Процесс продолжается до конца программы или вмешательства («СТОП») пользователя.

Интерфейс обучающей системы решает следующие задачи:

- вывод на экран структурной схемы;
- поддержку окон «Программа» и «Данные»;
- поддержку функциональных клавиш F2- F7;
- поддержку «Меню» для записи в память и исполнения команд программы;
- визуализацию процессов передачи, хранения и обработки данных и команд.

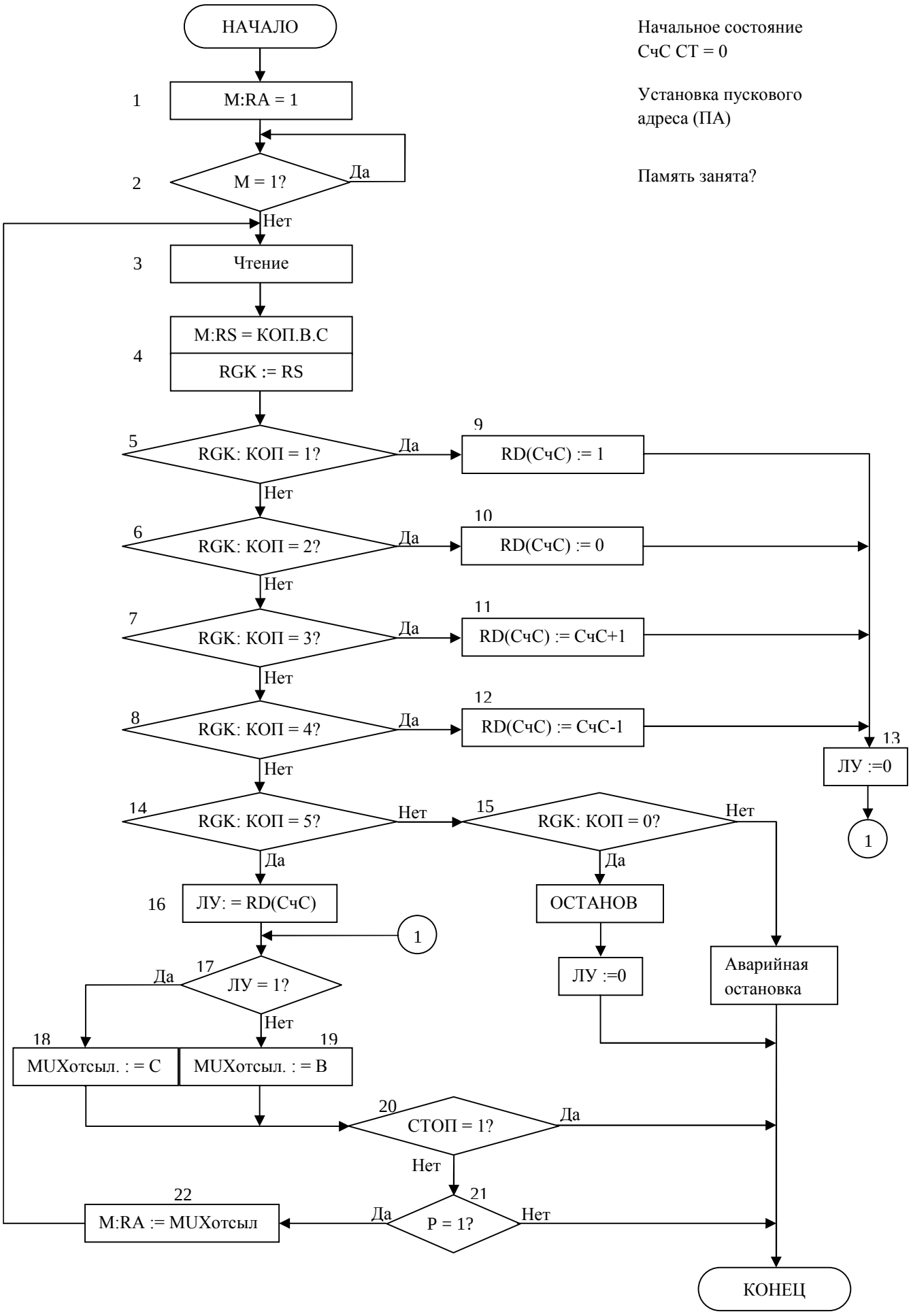


Рисунок 3 – Блок схема алгоритма работы машины Поста

8. Задание к лабораторной работе

- Исследовать структуру машины Поста;
- Изучить организацию шин адресов, данных и управления;
- Исследовать элементную базу процессора машины;
- Изучить поведение машины Поста по блок-схеме алгоритма ее работы;
- Составить алгоритм и программу, моделирующую работу логического элемента «2И». Пусть входные значения элемента будут находиться в ячейках RD0, RD1, а результат работы будет записываться в ячейку памяти RD2;
- Подготовить отчет по проделанной работе.

9. Содержание отчета

В отчете необходимо привести следующие сведения:

- Цель лабораторной работы;
- Структурную схему машины Поста;
- Алгоритм работы машины Поста;
- Описание структуры и алгоритма работы машины Поста;
- Алгоритм и программу, имитирующий работу логического элемента 2И;
- Выводы.

10. Контрольные вопросы

- Какой тип элемента памяти использован в регистре данных RD?
- Как организована запись и чтение разрядов регистра данных RD?
- Как согласована разрядность RD и счетчика CT?
- Как формируются адреса в шине (B, C)?
- Как организована передача данных из памяти RAM в регистр команд?
- Какой командой можно организовать условный переход?
- Как машина Поста исполнит команду, если ее код операции равен 7?
- Что такое ПДП и для чего он нужен?
- Что происходит во время выполнения сигнала СБРОС?

11. Список литературы

1. Прищепа Л. С. Архитектуры центральных процессорных устройств : Монография / Л. С. Прищепа. - Томск : В-Спектр, 2007. - 358[2] с. : ил., табл. - (Приоритетные национальные проекты. Образование). - Библиогр.: с. 354-355.
2. Безуглов Д. А. Цифровые устройства и микропроцессоры : учебное пособие для вузов / Д. А. Безуглов, И. В. Калиенко. - 2-е изд. - Ростов н/Д : Феникс, 2008. - 468, [12] с. : ил., табл. - (Высшее образование). - Библиогр.: с. 464-465

Лабораторная работа № 3

Содержание

Цель работы:

- Изучить систему команд микропроцессора (МП) семейства «Intel mcs-51».
- Изучить среду отладки программ для системы команд МП «edsim51».

1. Введение

Базис микропроцессора основан на логических схемах, которые спроектированы для работы с данными и выполнения вычислений. Поведение микропроцессора определяется исполняемой программой. Микропроцессор выполняет только две функции это – управление и обработка данных. Под термином «обработка» подразумевается перемещение данных и выполнение операций над ними. Термин «управление» определяет поведение аппаратных блоков процессора в определенный момент времени.

Работа микропроцессора состоит из нескольких шагов: сначала из памяти извлекается команда, затем логическая схема управления ее декодирует и синхронизирует работу исполнительных схем, тем самым исполняя эту команду. Эти шаги можно назвать циклом «выборка-исполнение». Для каждой команды, выполняется один такой цикл.

2. Система команд

Команда - это слово, которое извлекается микропроцессором из памяти программ, декодируется и им исполняется, таким образом, трансформируясь из информации в действие. Так, команды осуществляют пересылку данных, их обработку, а так же управляют аппаратными средствами микропроцессора во время всей его работы. Система команд определяет не только список команд, поддерживаемых микропроцессором, но и методы адресации к данным.

Команда состоит из двух частей: кода операции (КОП) и операнда. КОП – это идентификатор команды, при помощи которого микропроцессор дешифрирует информацию и преобразует ее в действие. Операнд это дополнительная информация, которая участвует в контексте выполнения команды и может содержать как обрабатываемые данные, так и адрес, по которому можно получить доступ к этим данным. Как правило, разрядность слова команды МП

совпадает с разрядностью слова данных, однако это правило не абсолютно. С целью оптимизации размера кристалла и сокращения энергопотребления МП, разработчики могут использовать неравную разрядность слов команд и слов данных. Команды могут иметь различную длину.

Для МП семейства Intel MCS-51 (далее МП MCS-51) размер команд составляет от одного до трех байт. Если длина команды составляет два или три байта, как показано на рисунке 1, то первое из них – это КОП, второе – адрес (старшая часть) / данные, третье – адрес (младшая часть).

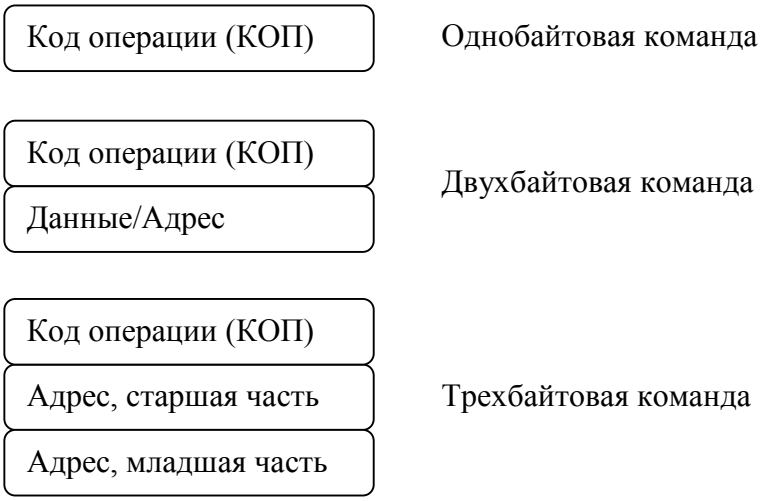


Рисунок 1 - Структура команд различной длины для МП MCS-51

Система команд МП MCS - 51 включает в себя 111 команд. Большинство команд выполняются за один или два машинных цикла «выборка-исполнение», за исключением команд умножения и деления, которые выполняются за четыре машинных цикла. В качестве операндов команд микропроцессора могут использоваться биты, четырехбитные цифры (ниблы), байты и двухбайтные слова.

По функциональным признакам команды разделяются на пять групп:

- пересылки данных;
- арифметических операций;
- логических операций;
- операций над битами;
- управления аппаратно-программными средствами МП.

В приложении 1 приведены таблицы, в которых отображаются названия команд, их мнемокод, производимую операцию, ее код, а так же размер команд в байтах и количество циклов, необходимых для исполнения этих команд. Ниже приведены обозначения, которые используются при описании команд.

Rn (n = 0,1,..., 7) - регистр общего назначения;
@Ri(i= 0, 1) - регистр общего назначения, работающий регистром косвенного адреса;
ad (address) - адрес прямо адресуемого байта;
ad11 - 11-разрядный абсолютный адрес перехода;
ad16 - 16-разрядный абсолютный адрес перехода;
rel - относительный адрес перехода;
#data8 - непосредственный операнд данных (1 байт);
#data16 - непосредственный операнд данных (2 байта);
bit - адрес прямо адресуемого бита;
/bit - инверсия прямо адресуемого бита;
A - аккумулятор;
PC (program counter)- счетчик команд;
DPTR (data pointer)- регистр указатель данных;
PSW (processor state word) – регистр состояния процессора (регистр флагов);
() - содержимое ячейки памяти или регистра.

2.1. Команды пересылки данных

Как показано в таблице 1, по команде MOV выполняется пересылка данных из второго операнда в первый. Эта команда не имеет доступа ни к внешней памяти данных, ни к памяти программ. Для этих целей используются команды M0VX и MOVC соответственно. Первая

из них обеспечивает чтение/запись байта из внешней памяти данных, вторая - чтение байта из памяти программ.

Таблица 1

Название команды	***Мнемокод***	***КОП***	***Б***	***Ц***	***Операция***
Пересылка в аккумулятор из регистра (n=0+7)	MOV A, R_n	1110.1rrr	1	1	(A)<-(R_n)
Пересылка в аккумулятор прямоадресуемого байта	MOV A, ad	1110.0101	2	1	(A)<-(ad)
Пересылка в аккумулятор байта из РПД (i=0,1)	MOV A, @R_i	1110.011i	1	1	(A)<-((R_i))
Загрузка в аккумулятор константы	MOV A, #data8	0111.0100	2	1	(A)<-#data8
Пересылка в регистр из аккумулятора	MOV R_n, A	1111.1rrr	1	1	(R_n)<-(A)
Пересылка в регистр прямоадресуемого байта	MOV R_n, ad	1010.1rrr	2	2	(R_n)<-(ad)
Загрузка в регистр константы	MOV R_n, #data8	0111.1rrr	2	1	(R_n)<-#data8
Пересылка по прямому адресу аккумулятора	MOV ad, A	1111.0101	2	1	(ad)<-(A)
Пересылка по прямому адресу регистра	MOV ad, R_n	1000.1rrr	2	2	(ad)<-(R_n)
Пересылка прямоадресуемого байта по прямому адресу	MOV add, ads	1000.0101	3	2	(add)<-(ads)
Пересылка байта из РПД по прямому адресу	MOV ad, @R_i	1000.011i	2	2	(ad)<-((R_i))
Пересылка по прямому адресу константы	MOV ad, #data8	0111.0101	3	2	(ad)<-#data8
Пересылка в РПД из аккумулятора	MOV @R_i, A	1111.011i	1	1	((R_i))<-(A)
Пересылка в РПД прямоадресуемого байта	MOV @R_i, ad	0110.011i	2	2	((R_i))<-(ad)
Пересылка в РПД константы	MOV @R_i, #data8	0111.011i	2	1	((R_i))<-#data8
Загрузка указателя данных	MOV DPTR, #data16	1001.0000	3	2	(DPTR)<-#data16
Пересылка в аккумулятор байта из ПП	MOVC A, @A+DPTR	1001.0011	1	2	A<-((A) +(DPTR))
Пересылка в аккумулятор байта из ПП	MOVC A, @A+PC	1000.0011	1	2	(PC)<-(PC)+1, (A)<-((A)+(PC))
Пересылка в аккумулятор байта из памяти данных	MOVX A, @Ri	1110.001i	1	2	(A)<-((Ri))
Пересылка в аккумулятор байта из расширенной памяти данных	MOVXA,@DPTR	1110.0000	1	2	(A)<-((DPTR))
Пересылка в память данных значение из аккумулятора	MOVX @R_i, A	1111.001i	1	2	((R_i))<-(A)
Пересылка в расширенную память данных значение из аккумулятора	MOVX @DPTR,A	1111.0000	1	2	((DPTR))<-(A)
Загрузка в стек	PUSH ad	1100.0000	2	2	(SP)<-(SP) + 1, ((SP))<-(ad)
Извлечение из стека	POP ad	1101.0000	2	2	(ad)<-(SP), (SP)<-(SP) – 1
Обмен аккумулятора с регистром	XCH A, R_n	1100.1rrr	1	1	(A)<->(Rn)
Обмен аккумулятора с прямоадресуемым байтом	XCH A, ad	1100.0101	2	1	(A)<->(ad)
Обмен аккумулятора с байтом из памяти данных	XCH A, @R_i	1100.011i	1	1	(A)<->((R_i))
Обмен младших тетрад аккумулятора и памяти данных	XCHD A, @R_i	1101.011i	1	1	(A_0...A_3)<->W(R_{i0}...R_{i3})

Команды XCH производят обмен данных, а команды PUSH и POP предназначены для записи данных в стек и их чтение. Размер стека ограничен размером памяти данных.

Группа команд пересылок микроконтроллера имеет следующую особенность - в ней нет специальных команд для работы со специальными регистрами: PSW, таймером, портами ввода-вывода. Доступ к ним, как и к другим регистрам специальных функций, осуществляется заданием соответствующего прямого адреса, т.е. это команды обычных пересылок, в которых вместо адреса можно ставить название соответствующего регистра. Например, чтение регистра флагов PSW в аккумулятор может быть выполнено командой «MOV A, PSW» которая преобразуется в машинный код следующим образом (при том, что адрес регистра PSW – D0h):

MOV A, D0h

E5D0,

где E5 - код операции, а D0 - операнд (адрес PSW).
Аккумулятор имеет два различных имени в зависимости от способа адресации: A - при неявной адресации (например, MOV A, R0) и ACC - при использовании прямого адреса.

2.2. Команды арифметических операций

Таблица 2

Название команды	***Мнемокод***	***КОП***	***Б***	***Ц***	***Операция***
Сложение аккумулятора с регистром (n= 0…7)	ADD A, R_n	0010.1rrr	1	1	(A)<-(A) + (R_n)
Сложение аккумулятора с прямоадресуемым байтом	ADD A, ad	0010.0101	2	1	(A)<-(A) + (ad)
Сложение аккумулятора с байтом из памяти данных (i = 0,1)	ADD A, @R_i	0010.011i	1	1	(A)<-(A) + ((R_i))
Сложение аккумулятора с константой	ADD A, #data8	0010.0100	2	1	(A)<-(A) + #data8
Сложение аккумулятора с регистром и переносом	ADDC A, R_n	0011.1rrr	1	1	(A)<-(A) + (R_n) + (C)
Сложение аккумулятора с прямоадресуемым байтом и переносом	ADDC A, ad	0011.0101	2	1	(A)<-(A) + (ad) + (C)
Сложение аккумулятора с байтом из памяти данных и переносом	ADDC A, @R_i	0011.011i	1	1	(A)<-(A) + ((R_i)) + (C)
Сложение аккумулятора с константой и переносом	ADDC A, #data8	0011.0100	2	1	(A)<-(A) + #data8 + (C)
Десятичная коррекция аккумулятора	DAA	1101.0100	1	1	Если (A_0…A_3)>9 или ((AC)=1), то (A_0…A_3)<-(A_0…A_3) + 6, затем если (A_4…A_7)>9 или ((C)=1), то (A_4…A_7) <- (A_4…A_7) + 6
Вычитание из аккумулятора регистра и заёма	SUBB A, R_n	1001.1rrr	1	1	(A)<-(A) – (C) – (R_n)
Вычитание из аккумулятора прямоадресуемого байта и заема	SUBB A, ad	1001.0101	2	1	(A)<-(A) – (C) – ((ad))
Вычитание из аккумулятора байта памяти данных и заема	SUBB A, @R_i	1001.011i	1	1	(A)<-(A) – (C) – ((R_i))
Вычитание из аккумулятора константы и заема	SUBB A, data8	1001.0100	2	1	(A)<-(A) – (C) - #data8
Инкремент аккумулятора	INC A	0000.0100	1	1	(A)<-(A) + 1
Инкремент регистра	INC R_n	0000.1rrr	1	1	(R_n)<-(R_n)+ 1
Инкремент прямоадресуемого байта	INC ad	0000.0101	2	1	(ad)<-(ad) + 1
Инкремент байта в памяти данных	INC @R_i	0000.011i	1	1	((R_i))<-((R_i))+1
Инкремент указателя данных	INC DPTR	1010.0011	1	2	(DPTR)<-(DPTR) + 1
Декремент аккумулятора	DEC A	0001.0100	1	1	(A)«-(A)-1
Декремент регистра	DEC R_n	0001.1rrr	1	1	(R_n)<-(R_n)-1
Декремент прямоадресуемого байта	DEC ad	0001.0101	2	1	(ad)<-(ad)-1
Декремент байта в памяти данных	DEC @R_i	0001.011i	1	1	«R_i))<-((R_i))-1
Умножение аккумулятора на регистр B	MUL AB	1010.0100	1	4	(B)(A)<-(A)*(B)
Деление аккумулятора на регистр B	DIV AB	1000.0100	1	4	(A).(B)<-(A)/(B)

В таблице 2 приведен список арифметических операций, поддерживаемых микропроцессором. Можно заметить, что результат выполнения команд ADD, ADDC, SUBB, MUL и DIV отображается флагами регистра PSW. Флаг C (carry) устанавливается при переносе из разряда D7, т. е. в случае, если результат не помещается в восемь разрядов; флаг AC (advanced carry)устанавливается при переносе из разряда D3 в командах сложения и вычитания и служит для реализации десятичной арифметики. Этот признак используется командой DAA.

Флаг OV устанавливается при переносе из разряда D6, т. е. в случае, если результат не помещается в семь разрядов и восьмой не может быть интерпретирован как знаковый. Этот признак служит для организации обработки знаковых чисел.
Флаг P устанавливается и сбрасывается аппаратно. Если число единичных бит в аккумуляторе нечетно, то P = 1, в противном случае P = 0.

2.3. Команды логических операций

Таблица 3

Название команды	***Мнемокод***	***КОП***	***Б***	***Ц***	***Операция***
Логическое И аккумулятора и регистра	ANL A, R_n	0101.1rrr	1	1	(A)<-(A) AND (R_n)
Логическое И аккумулятора и прямоадресуемого байта	ANL A, ad	0101.0101	2	1	(A)<-(A) AND (ad)
Логическое И аккумулятора и байта из памяти данных	ANL A, @R_i	0101.011i	1	1	(A)<-(A)AND((R_i))
Логическое И аккумулятора и константы	ANL A, #data8	0101.0100	2	1	(A)<-(A) AND #data8
Логическое И прямоадресуемого байта и аккумулятора	ANL ad, A	0101.0010	2	1	(ad)<-(ad) AND (A)
Логическое И прямоадресуемого байта и константы	ANL ad, #data8	0101.0011	3	2	(ad)<-(ad) AND #data8
Логическое ИЛИ аккумулятора и регистра	ORLA, R_n	0100.1rrr	1	1	(A)<-(A) OR (R_n)
Логическое ИЛИ аккумулятора и прямоадресуемого байта	ORLA, ad	0100.0101	2	1	(A)<-(A) OR (ad)
Логическое ИЛИ аккумулятора и байта из памяти данных	ORLA, @R_i	0100.011i	1	1	(A)<-(A) OR ((R_i))
Логическое ИЛИ аккумулятора и константы	ORL A, #data8	0100.0100	2	1	(A)<-(A) OR #data8
Логическое ИЛИ прямоадресуемого байта и аккумулятора	ORL ad, A	0100.0010	2	1	(ad)<-(ad) OR (A)
Логическое ИЛИ прямоадресуемого байта и константы	ORL ad, #data8	0100.0011	3	2	(ad)<-(ad) OR #data8
Исключающее ИЛИ аккумулятора и регистра	XRL A, R_n	0110.1rrr	1	1	(A)<-(A) XOR (R_n)
Исключающее ИЛИ аккумулятора и прямоадресуемого байта	XRL A, ad	0110.0101	2	1	(A)<-(A) XOR (ad)
Исключающее ИЛИ аккумулятора и байта памяти данных	XRL A, @R_i	0110.0111	1	1	($_i$A)<-(A) XOR ((R_i))
Исключающее ИЛИ аккумулятора и константы	XRL A, #data8	0110.0100	2	1	(A)<-(A) XOR #data8
Исключающее ИЛИ прямоадресуемого байта и аккумулятора	XRL ad, A	0110.0010	2	1	(ad)<-(ad) XOR (A)
Исключающее ИЛИ прямоадресуемого байта и константы	XRL ad, #data8	0110.0011	3	2	(ad)<-(ad) XOR #data8
Сброс аккумулятора	CLR A	1110.0100	1	1	(A)<-0
Инверсия аккумулятора	CPL A	1111.0100	1	1	(A)<-NOT(A)
Сдвиг аккумулятора влево циклический	RL A	0010.0011	1	1	(A_{n+1})<-(A_n), n=0…6, (A_0) <- (A_7)
Сдвиг аккумулятора влево через перенос	RLC A	0011.0011	1	1	(A_{n+i})<-(A_n), n=0…6, (A_0) <- (C), (C) <-A_7)
Сдвиг аккумулятора вправо циклический	RR A	0000.0011	1	1	(A_n)<-(A_{n+}i), n=0…6, (A_7)<- (A_0)
Сдвиг аккумулятора вправо через перенос	RRC A	0001.0011	1	1	(A_n)<-(A_{n+}i), n=0…6, (A_7) <- (C), (C) <-(A_0)
Обмен местами тетрад в аккумуляторе	SWAP A	1100.0100	1	1	(A_0...A_3)<->(A_4...A_7)

В таблице 3 дано краткое описание команд логических операций, которые выполняют следующие преобразования над байтами: логическое «И», логическое «ИЛИ», «исключающее ИЛИ», инверсия, сброс в нулевое значение и сдвиг значения, которое хранится в аккумуляторе влево или вправо.

2.4. Команды битовых операций

Группа состоит из 12 команд, краткое описание которых приведено в таблице 4. Эти команды позволяют выполнять операции над отдельными битами: сброс, установку, инверсию бита, а также логические «И» и «ИЛИ.

Таблица 4

Название команды	*Мнемокод*	*КОП*	*Б*	*Ц*	*Операция*
Сброс переноса	CLRC	1100.0011	1	1	(C)<-0
Сброс бита	CLR bit	1100.0010	2	1	(bit)<-0
Установка переноса	SETBC	1101.0011	1	1	(C)<-1
Установка бита	SETB bit	1101.0010	2	1	(bit)<-1
Инверсия переноса	CPLC	1011.0011	1	1	(C)<-NOT(C)
Инверсия бита	CPL bit	1011.0010	2	1	(bit)<-NOT(bit)
Логическое И бита и переноса	ANL C, bit	1000.0010	2	2	(C)<-(C) AND (bit)
Логическое И инверсии бита и переноса	ANL C, /bit	1011.0000	2	2	(C)<-(C)AND (NOT(b))
Логическое ИЛИ бита и переноса	ORL C, bit	0111.0010	2	2	(C)<-(C) OR (bit)
Логическое ИЛИ инверсии бита и переноса	ORL C, /bit	1010.0000	2	2	(C)<-(C)OR(NOT(bit))
Пересылка бита в перенос	MOV C, bit	1010.0010	2	1	(C)<-(bit)
Пересылка переноса в бит	MOV bit, C	1001.0010	2	2	(bit)<-(C)

В качестве "логического" аккумулятора, участвующего во всех операциях с двумя операндами, выступает флаг переноса «С» (разряд D7 PSW), а в качестве операндов могут использоваться 128 бит памяти данных и регистры специальных функций, допускающие адресацию отдельных бит.

2.5. Команды управления ресурсами МП

Группа представлена командами безусловного и условного переходов, командами вызова подпрограмм и командами возврата из подпрограмм, краткое описание которых приведено в таблице 5.

Команда безусловного перехода LJMP (long jump) осуществляет переход по абсолютному 16-битному адресу, указанному в теле команды, т. е. команда обеспечивает переход в любую точку памяти программ.

Действие команды AJMP (absolute jump) аналогично команде LJMP, однако в команде указаны лишь 11 младших разрядов адреса. Поэтому переход осуществляется в пределах страницы размером 2 Кбайт.

В отличие от предыдущих команд, в команде SJMP (short jump) указан не абсолютный, а относительный адрес перехода. Величина смещения rel рассматривается как число со знаком, и следовательно, переход возможен в пределах- 128...+127 байт относительно адреса команды, следующей за командой SJMP.

Таблица 5

Название команды	*Мнемокод*	*КОП*	*Б*	*Ц*	*Операция*
Длинный переход в полном объеме памяти программ	LJMP ad16	0000.0010	3	2	(PC)<-ad16
Абсолютный переход внутри страницы памяти	AJMP ad11	$a_{10}a_9a_8$0.0001	2	2	(PC)<-(PC)+2, (PC_{0-10})<-ad11
Короткий относительный переход	SJMP rel	1000.0000	2	2	(PC)<-(PC)+2, (PC)<-(PC)+rel
Косвенный относительный переход	JMP @A+DPTR	0111.0011	1	2	(PC)<-(A)+(DPTR)
Переход, если аккумулятор равен нулю	JZ rel	0110.0000	2	2	(PC)<-(PC)+2, если (A)=0, то (PC)<-(PC)+rel
Переход, если аккумулятор не равен нулю	JNZ rel	0111.0000	2	2	(PC)<-(PC)+2, если (A)*0, то (PC)<-(PC)+rel
Переход, если перенос равен единице	JC rel	0100.0000	2	2	(PC)<-(PC)+2, если (C)=1, то (PC)<-(PC)+rel
Переход, если перенос равен нулю	JNC rel	0101.0000	2	2	(PC)<-(PC)+2, если (C)=0, то (PC)<-(PC)+rel
Переход, если бит равен единице	JB bit, rel	0010.0000	3	2	(PC)<-(PC)+3, если (bit)=1, то (PC)<-(PC)+rel
Переход, если бит равен нулю	JNB bit, rel	0011.0000	3	2	(PC)<-(PC)+3, если (bit)=0, то (PC)<-(PC)+rel
Переход, если бит установлен, с последующим сбросом бита	JBC bit, rel	0001.0000	3	2	(PC)<-(PC)+3, если(bit)=1, то (bit)<-0 и (PC)<-(PC)+rel
Декремент регистра и переход, если не нуль	DJNZ Rn, rel	1101.1rrr	2	2	(PC)<-(PC)+2, (R_n)<-(R_n)-1, если (R_n)<>0, то (PC)<-(PC)+rel
Декремент прямоадресуемого байта и переход, если не нуль	DJNZ ad, rel	1101.0101	3	2	(PC)<-(PC)+2, (ad)<-(ad)-1, если (ad)<>0, то (PC)<-(PC)+rel
Сравнение аккумулятора с прямоадресуемым байтом и переход, если не равно	CJNE A, ad, rel	1011.0101	3	2	(PC)<-(PC)+3.если (A)<>(ad), то (PC)<-(PC)+re1,если (A)<(ad), то (C)<-1, иначе (C)<-0
Сравнение аккумулятора с константой и переход, если не равно	CJNE A, #data8, rel	1011.0100	3	2	(PC)<-(PC)+3.если (A)*#d8, то (PC)<-(PC)+rel, если (A)<#d8, то (C)<-1, иначе (C)<-0
Сравнение регистра с константой и переход, если не равно	CJNE R_n, #data8, rel	1011.1rrr	3	2	(PC)<-(PC)+3.если (Rn)<>#d, то (PC)<-(PC)+rel, если (Rn) <#d, то (C)<-1, иначе (C)<-0
Сравнение байта с константой и переход, если не равно	CJNE @R_i,#data8, rel	1011.011i	3	2	(PC)<-(PC)+3.если ((Ri))*#d, то (PC)<-(PC)+re1,если ((Ri)) <#d, то (C)<-1, иначе (C)<-0
Длинный вызов подпрограммы	LCALL ad16	0001.0010	3	2	(PC)<-(PC)+3, (SP)<-(SP)+1, ((SP))<-($PC_{0...7}$), (SP)<-(SP)+1, ((SP))<-(PC_{8-15}), (PC)<-ad16
Абсолютный вызов подпрограммы в пределах страницы	ACALL ad11	$a_{10}a_9a_8$1.0001	2	2	(PC)<-(PC)+2, (SP)<-(SP)+1, ((SP))<-($PC_{0...7}$), (SP)<-(SP)+1, ((SP))<-($PC_{8...15}$), (PC0-10)<-ad11
Возврат из подпрограммы	RET	0010.0010	1	2	($PC_{8...15}$)<-((SP)), (SP)<-(SP)-1, ($PC_{0...7}$) -((SP)), (SP)<- SP)-1
Возврат из подпрограммы обработки прерывания	RETI	0011.0010	1	2	($PC_{8...15}$) <-((SP)), (SP)<-(SP)-1, ($PC_{0...7}$) <- ((SP)), (SP)<-(SP)-1
Пустая операция	NOP	0000.0000	1	1	(PC)<-(PC)+1

Команда косвенного перехода JMP @A+DPTR позволяет вычислять адрес перехода в процессе выполнения самой программы.

Командами условного перехода можно проверять следующие условия:

JZ (jump if zero) — аккумулятор содержит нулевое значение;

JNZ (jump if not zero) — аккумулятор содержит не нулевое значение;

JC (jump if carry) — бит переноса C равен 1;

JNC (jump if not carry)— бит переноса C равен 0;

JB (jump if bit)— прямо адресуемый бит равен 1

JNB(jump if not bit) — прямо адресуемый бит равен 0;

JBC (jump if bit and clear)— прямо адресуемый бит равен 1 и сбрасывается в нулевое значение при выполнении команды.

Все команды условного перехода , как и команда SJMP содержат короткий относительный адрес, т. е. переход может осуществляться в пределах—128... +127 байт относительно следующей команды.

Команда DJNZ (decrement jump if not zero) предназначена для организации программных циклов. Регистр R_n или байт по адресу ad,

указанные в теле команды, содержат счетчик повторений цикла, а смещение rel — относительный адрес перехода к началу цикла. При выполнении команды содержимое счетчика уменьшается на 1 и проверяется на 0. Если значение содержимого счетчика не равно 0, то осуществляется переход на начало цикла, в противном случае выполняется следующая команда.

Команда CJNE (compare jump if not equal) для реализации процедур ожидания внешних событий. В теле команды указаны "координаты" двух байт и относительный адрес перехода rel. В качестве двух байт могут быть использованы, например, значения содержимого аккумулятора и прямо адресуемого байта или косвенно адресуемого байта и константы. При выполнении команды значения указанных двух байт сравниваются и в случае, если они не одинаковы, осуществляется переход. Например, команда

WAIT: CJNE A, P0, WAIT

будет выполняться до тех пор, пока значения на линиях порта P0 не совпадут со значениями содержимого аккумулятора.

Действие команд вызова процедур полностью аналогично действию команд безусловного перехода. Единственное отличие состоит в том, что они сохраняют в стеке адрес возврата.

Команда возврата из подпрограммы RET восстанавливает из стека значение счетчика команд, а команда возврата из процедуры обработки прерывания RETI, кроме того, разрешает прерывание. Команды LCALL ACALL вызывают подпрограмму, то есть это команды безусловного перехода, при котором в стеке сохраняется полный 16-разрядный адрес возврата из подпрограммы и модифицируется счетчик команд новым значением. Это значение определяет адрес следующей исполняемой команды. Различие этих команд в том, что ACALL (absolute call) обеспечивает переход в пределах одной страницы памяти, размер которой составляет 2048 байт, тогда как LCALL (long call) обеспечивает длинный переход по всему адресному пространству программной памяти, что составляет 65536 байт.

3. Методы адресации

Набор команд МП MCS-51 поддерживает следующие методы адресации:

- *Прямая* адресация (Direct Addressing).Операнд определяется 8-битным адресом в инструкции. Эта адресация используется только для внутренней памяти данных и регистров.
- *Косвенная* адресация (Indirect Addressing). Адрес регистра, содержащий адрес операнда, описан самой инструкцией и содержится в коде операции. Данный вид адресации может применяться при обращении как к внутренней, так и внешней памяти. Для указания 8-битных адресов могут использоваться регистры R0 и R1 или указатель стека SP. Для 16-битной адресации используется только регистр (DPTR - Data Pointer - "указатель данных").
- *Регистровая* адресация (Register Instruction).Данная адресация применяется для доступа к регистрам R0…R7. Команды с регистровой адресацией содержат в коде операции трехбитовое поле, которое определяет номер регистра.
- *Непосредственная* адресация (Immediate constants).Операнд содержится в теле команды и следует за кодом операции. Размер операнда составляет один или два байта в котором содержится константа (#data8, #data16).
- *Индексная* адресация (Indexed Addressing). Индексная адресация используется при чтении памяти программ. В этом режиме осуществляется просмотр таблиц в памяти программ. 16-битовый регистр (DPTR или PC) содержит базовый адрес требуемой таблицы, а аккумулятор содержит индекс, т.е. указывает на порядковый номер элемента таблицы. Адрес элемента таблицы находится сложением базы с индексом (содержимым аккумулятора).
- *Неявная* адресация (Register-Specific Instructions).Название «неявная адресация» подразумевает, что адрес к регистрам или памяти не указывается при помощи операнда. Адрес регистра определяется самой инструкцией и содержится в коде операции. Например, некоторые инструкции используют индивидуальные регистры, такие, как аккумулятор, или DPTR, но при этом, адрес этих регистров не объявлен в операнде, так как он уже определен командой и определяется микропроцессором из кода операции. Примером неявной адресации к памяти могут являться команды push, pop.

4. Регистры специальных функций

Регистры специальных функций управляют аппаратно-программными блоками, которые входят в состав микропроцессора семейства Intel mcs-51. В адресное пространство памяти данных входит адресное пространство регистров специальных функций SFR (Special Function Register). В таблице 6 показано размещение регистров специальных функций.

Регистры, символ которых отмечен знаком «*», допускают адресацию отдельных бит при использовании команд из группы команд битовых операций.

Регистры занимают только часть 128-байтового пространства. Те ячейки памяти с адресами 80H-0FFH, которые не заняты регистрами, физически отсутствуют, при чтении данных по эти адресам можно получить лишь код команды возврата.

Регистры-защелки SFR параллельных портов P0...P3 - служат для ввода-вывода информации.

Две регистровые пары с именами THO, TL0 (Timer High 0 и Timer Low 0 соответственно) и TH1, TL1 представляют собой 16-битные регистры, которые управляют работой двух таймеров-счетчиков. Режимы работы этих устройств задаются с использованием регистра TMOD, а управление ими осуществляется с помощью регистра TCON.

Регистр PCON используется для управления режимами энергопотребления.

Таблица 6

Адрес (hex)	Символ	Наименование
E0	*ACC	Аккумулятор (Accumulator)
F0	*B	Регистр расширитель аккумулятора (Multiplication Register)
D0	*PSW	Слово состояния программы (Program Status Word)
80h	*P0	Порт 0 (SFR P0)
90h	*P1	Порт 1 (SFR P1)
A0	*P2	Порт 2 (SFR P2)
B0	*P3	Порт 3 (SFR P3)
81	SP	Регистр указатель стека (Stack Pointer)
83	DPH	Старший байт регистра указателя данных DPTR (Data Pointer High)
82	DPL	Младший байт регистра указателя данных DPTR (Data Pointer Low)
8C	THO	Старший байт таймера 0
8A	TLO	Младший байт таймера 0
8D	TH1	Старший байт таймера 1
8B	TL1	Младший байт таймера 1

89	TMOD	Регистр режимов таймеров счетчиков (Timer/Counter Mode Control Register)
88	*TCON	Регистр управления статуса таймеров (Timer/Counter Control Register)
B8	*IP	Регистр приоритетов (Interrupt Priority Control Register)
A8	*IE	Регистр маски прерывания (Interrupt Enable Register)
87	PCON	Регистр управления мощностью (Power Control Register)
98	*SCON	Регистр управления приемопередатчиком (Serial Port Control Register)
99	SBUF	Буфер приемопередатчика (Serial Data Buffer)

Регистры IP и IE управляют работой системы прерываний.

Регистры SBUF и SCON управляют работой приемопередатчика последовательного порта.

Восьми битный регистр-указатель стека SP может адресовать любую область внутренней памяти данных.

Регистр-указатель данных DPTR используется для фиксации 16-битного адреса в операциях обращения к внешней памяти.

Аккумулятор (ACC) является источником операнда и местом фиксации результата при выполнении арифметических, логических операций и ряда операций передачи данных. При помощи аккумулятора могут быть выполнены операции сдвигов, проверка на нуль, формирование флага паритета и т.п.

Регистр B используется как источник и как приемник при операциях умножения и деления, обращение к нему, как к регистру SFR, производится аналогично аккумулятору.

При выполнении многих команд АЛУ формирует ряд признаков операции, которые фиксируются в регистре флагов PSW (processor state word).

5. Регистр флагов (PSW)

В таблице 7 приведен перечень флагов, их символические имена и условия формирования. Наиболее "активным" флагом PSW является флаг переноса, который принимает участие и модифицируется в процессе выполнения множества операций, включая сложение, вычитание и сдвиги. Кроме того, флаг переноса (C) выполняет функции "булева аккумулятора" в командах, манипулирующих битами. Флаг переполнения (OV) фиксирует арифметическое переполнение при операциях над целыми числами со знаком и делает возможным использование арифметики в дополнительных кодах. Значение битов выбора банков регистров (RSO, RS1) определяется прикладной

программой и используется для выбора одного из четырёх регистровых банков. Программист может использовать регистры в различных банках при работе с подпрограммами. В тех случаях, когда перед вызовом подпрограммы, значения, хранящиеся в регистрах R0-R7 будут еще использоваться, вместо того, чтобы сохранять их в стеке, можно переключить другой банк регистров и работать в нем. При возврате из подпрограммы, вместо того, чтобы загружать из стека сохраненные данные, достаточно снова переключиться на начальный банк и продолжить выполнение основной программы.

Таблица 7

<table>
<tr><th>Символ</th><th>Позиция</th><th>Имя и назначение</th></tr>
<tr><td>P</td><td>PSW.0</td><td>Флаг паритета. Устанавливается и сбрасывается на аппаратном уровне в каждом цикле команды и фиксирует нечетное/четное число единичных бит в аккумуляторе</td></tr>
<tr><td>-</td><td>PSW.1</td><td>Не используется</td></tr>
<tr><td>OV</td><td>PSW.2</td><td>Флаг переполнения. Устанавливается и сбрасывается на аппаратном уровне при выполнении арифметических операций</td></tr>
<tr><td>RS0
RS1</td><td>PSW.3
PSW.4</td><td>Биты выбора банка регистров.
<table>
<tr><th>RS0 RS1</th><th>Банк</th><th>Границы адресов ОЗУ</th></tr>
<tr><td>00</td><td>0</td><td>00H - 07H</td></tr>
<tr><td>10</td><td>1</td><td>08H - 0FH</td></tr>
<tr><td>01</td><td>2</td><td>10H-17H</td></tr>
<tr><td>11</td><td>3</td><td>18H-1FH</td></tr>
</table>
</td></tr>
<tr><td>F0</td><td>PSW.5</td><td>Флаг пользователя. Может быть установлен, сброшен или проверен программно.</td></tr>
<tr><td>AC</td><td>PSW.6</td><td>Флаг вспомогательного переноса. Устанавливается и сбрасывается только аппаратными средствами. При выполнении команд сложения и вычитания сигнализирует о переносе в бите 3 аккумулятора (ACC).</td></tr>
<tr><td>C</td><td>PSW.7</td><td>Флаг переноса. Устанавливается и сбрасывается как аппаратно, так и программно.</td></tr>
</table>

6. Среда отладки и симуляции программ «edsim51»

Симулятор «edsim51» предназначен для написания и отладки программ для микропроцессоров семейства intel mcs-51. Программирование процессора осуществляется при помощи построения программы на языке ассемблер. Внешний вид симулятора приведен на рисунке 2.

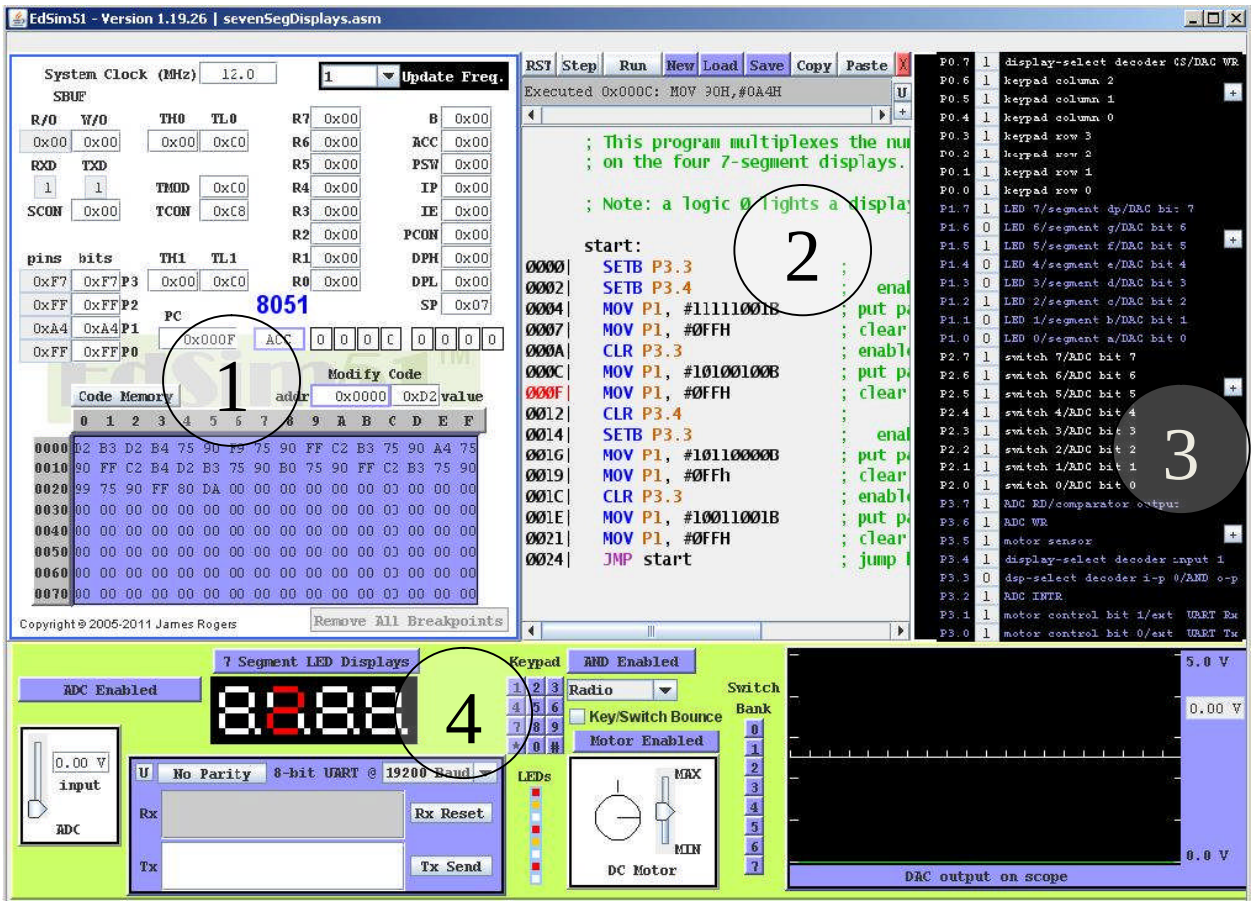


Рисунок 2 - Рабочее окно симулятора «edsim51»

Как видно, рабочее окно симулятора состоит из четырех частей:

1. Область памяти микропроцессора, в которой отображаются значения регистров общего назначения, регистров специальных функций. Показан дамп памяти программ и памяти данных. В этой области отображается элементы управления при отладки программы (System clock, Update frequency, Remove All Breakpoints).
2. Окно пользователя. В этом окне отображается отлаживаемый код, который исполняется в этой микропроцессорной среде.
3. Окно портов ввода/вывода предназначено для детального отображения состояния значений портов P0-P3 в бинарном виде. Текстом подписаны устройства, которые подключены к соответствующим портам ввода/вывода.

4. Панель внешних устройств ввода - вывода, подключенных к микропроцессору. Состоит из следующих устройств (слева - направо, сверху - вниз):

- Аналого-цифровой преобразователь (АЦП), который преобразует аналоговое напряжение в цифровой вид (ADC);
- Дисплей, который состоит из четырех семисегментных индикаторов, подключенных по схеме динамической индикации (7 Segment LED Displays);
- Приемо-передатчик последовательного порта (8-bit UART);
- Клавиатурный модуль (Keypad);
- Модуль светодиодной индикации (LEDs);
- Имитатор вращения двигателя постоянного тока (DC Motor);
- Модуль переключателей (Switch bank);
- Цифро-аналоговый преобразователь (ЦАП), который формирует аналоговое напряжение из цифрового кода. Выход ЦАП подключен к имитатору осциллографа, который динамически отображает форму и уровень сформированного напряжения. (DAC output on scope).

Схема подключения этих устройств и модулей к микропроцессору приведена в приложении 1. Благодаря этой схеме можно определить как, к какому порту подключено устройство и как организовано взаимодействие с микропроцессором.

7. Характеристики и описание микропроцессора 8051

Микропроцессор 8051, семейства Intel mcs – 51 содержит:

- 4 КБ памяти программ;
- 128 байт памяти данных;
- 2 таймера;
- Четыре восьмибитных порта ввода / вывода (P0-P3);
- Интерфейс последовательного порта;
- Адресное пространство памяти программ составляет 64 Кб;
- Адресное пространство памяти данных составляет 64 Кб;
- Процессор логических операций (Булева логика);
- Умножитель/делитель.

На рисунке 3 представлено условно графическое изображение, которое отображают номера и наименование выводов процессора.

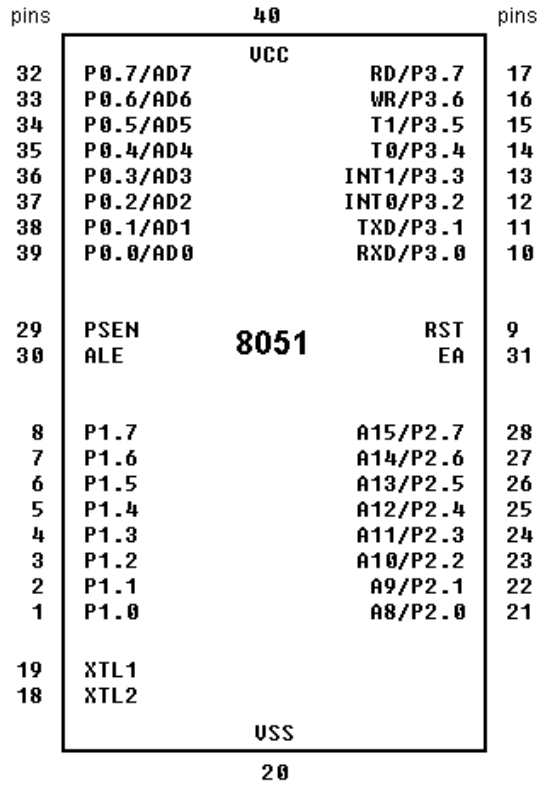


Рисунок 3 - Условно графическое изображение микропроцессора 8051

P0 - порт ввода вывода общего назначения, в альтернативном режиме применяется для подключения мультиплексированной шины адреса/данных внешней памяти,

P1 - используется как порт ввода/вывода и в альтернативном режиме не используется,

P2 - в альтернативном режиме предназначен для организации адресной шины при подключении микропроцессора к внешней памяти,

P3 – как и три предыдущих порта, в работает в двух режимах, обычный, при котором порт работает в общем режиме (управляется программно при помощи регистра P3), и альтернативным, в котором управление осуществляется аппаратными ресурсами самого микропроцессора. Этот порт в альтернативном режиме предназначен для организации шины управления внешними аппаратными устройствами ввода/вывода. Краткое описание приведено в таблице 8.

Таблица 8

Общее название вывода	Альтернативное название вывода	Описание
P3.0	RXD	Приемная линия последовательного порта
P3.1	TXD	Передающая линия последовательного порта
P3.2	INT0	Вход внешнего прерывания INT0
P3.3	INT1	Вход внешнего прерывания INT1
P3.4	T0	Вход внешнего управления Таймером 0
P3.5	T1	Вход внешнего управления Таймером 1
P3.6	WR	Управляющий сигнал записи во внешнюю память
P3.7	RD	Управляющий сигнал чтения из внешней памяти

ALE (address latch enable) – сигнал управления внешней памяти. При помощи этого сигнала внешняя память определяет какая информация передается по мультиплексированной шине адрес/данные. Таким образом, при ALE = «1», по шине передается адрес, при ALE = «0», передаются данные.

PSEN (Program store enable) – управляющий сигнал, при помощи которого процессор синхронизует работу с памятью программ и данных, т.е. при PSEN = «1», шина адресов/данных используется в работе с памятью программ, при PSEN = «0», с памятью данных.

XTL1, XTL2 – входы, которые используются для подключения схемы генератора синхроимпульсов.

8. Запуск среды отладки и открытие примера программы.

Для того, чтобы запустить среду отладки надо открыть папку «edsim51» и в корне запустить двойным нажатием файл «edsim51.jar», после этого можно увидеть экранную форму, вид который представлен на рисунке 2.

Чтобы открыть рабочий пример программы надо воспользоваться кнопкой «Load», которые расположены в верхней части окна пользователя. После этого, выбрать пример программы, выделив одинарным нажатием мыши и подтвердив нажатием кнопки «Open», как показано на рисунке 4.

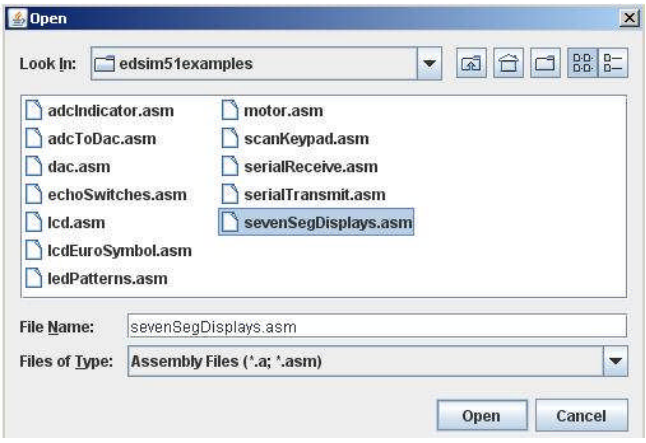


Рисунок 4 - Внешний вид диалогового окна открытия кода программы

При помощи кнопки «RST», расположенной в верхней части окна пользователя можно произвести начальную инициализацию имитируемого процессора, при этом обнулится значение счетчика команд и регистров. Значение указателя стека при этом, будет равно 0x07. Выполнение программы возможно в автоматическом и пошаговом режиме. Пошаговый режим применяется для отладки программы и активируется нажатием кнопки «Assm», для продолжения выполнения программы по шагам необходимо нажимать кнопку «Step». Автоматический режим позволяет пользователю запустить программу на исполнение, при этом останов программы возможен в местах, в которых установлена точка останова программы (breakpoint). Точка останова программы может быть выставлена путем двойного нажатия левой кнопки мыши на адрес инструкции, как показано на рисунке 5.

```
000C|   MOV P1, #10100100B       ; put pattern for 2 on display
000F|   MOV P1, #0FFH           ; clear the display
0012|   CLR P3.4                ; |
0014|   SETB P3.3                ; | enable display 1
0016|   MOV P1, #10110000B       ; put pattern for 3 on display
0019*   MOV P1, #0FFh           ; clear the display
001C|   CLR P3.3                ; enable display 0
001E|   MOV P1, #10011001B       ; put pattern for 4 on display
0021*   MOV P1, #0FFH           ; clear display
0024|   JMP start                ; jump back to start
```

Рисунок 5 - Отображение точек останова программы (адреса 0x0019 и 0x0021)

При помощи кнопки «New» пользователь может создать новый файл программы, а при помощи кнопки «Save», сохранить файл программы на диск.

9. Варианты заданий к лабораторной работе

9.1. Используя систему команд процессора 8051, составьте алгоритм и напишите программу циклического перебора значения от 50 до 99 включительно. Значения выводить в порт P1. Оцените период выполнения цикла. Проверьте правильность работы программы на симуляторе.

9.2. Используя систему команд процессора 8051, составьте алгоритм и напишите программу циклического перебора значения от 99 до 0 включительно. Нечетные значения выводить в порт P1, четные в P0. Оцените период выполнения цикла. Проверьте правильность работы программы на симуляторе.

9.3. Используя систему команд процессора 8051, составьте алгоритм и напишите программу циклического перебора значения от 99 до 0. Единицы выводить в порт P0, десятки в порт P1. Оцените период выполнения цикла. Проверьте правильность работы программы на симуляторе.

9.4. Используя систему команд процессора 8051, составьте алгоритм и напишите программу циклического перебора значения от 0 до 100 включительно. Нечетные значения выводить в порт P0, четные в P1. Оцените период выполнения цикла. Проверьте правильность работы программы на симуляторе.

9.5. Используя систему команд процессора 8051, составьте алгоритм и напишите программу циклического перебора значения от 0 до 99. Единицы выводить в порт P1, десятки в порт P0. Оцените период выполнения цикла. Проверьте правильность работы программы на симуляторе.

9.6. Используя систему команд процессора 8051 и схему моделируемой среды, составьте алгоритм и напишите программу поочередного зажигания светодиодов LED0-LED7. Оцените период выполнения цикла. Проверьте правильность работы программы на симуляторе.

9.7. Используя систему команд процессора 8051 и схему моделируемой среды, составьте алгоритм и напишите программу управляемого зажигания светодиодов LED0-LED7 при помощи

выключателей SW0-SW7. Оцените период выполнения цикла. Проверьте правильность работы программы на симуляторе.

10. Содержание отчета

В отчете необходимо привести следующие сведения:

- Цель лабораторной работы;
- Описание системы команд
- Описание среды отладки и симуляции программ «edsim51»;
- Вариант решения задания (алгоритм, код);
- Выводы.

11. Контрольные вопросы

- Какие команды МП MCS-51 предназначены для передачи данных?
- Какие команды МП MCS-51 предназначены для организации подпрограмм?
- Какие методы адресации используются в вашей программе?
- Как можно проверить нулевой результат выполнения арифметической операции?
- При помощи каких команд можно организовать цикл?

12. Список литературы

1. Шарапов А. В. Проектирование микропроцессорных устройств : руководство к выполнению курсовых проектов (в том числе ГПО) для студентов специальности "Промышленная электроника" / А. В. Шарапов ; Федеральное агентство по образованию, Томский государственный университет систем управления и радиоэлектроники, Кафедра промышленной электроники. - Томск : ТУСУР, 2009. - 74 с. : ил. - Библиогр.: с. 74
2. Калабеков Б. А. Цифровые устройства и микропроцессорные системы : Учебник для средних специальных учебных заведений связи / Б. А. Калабеков. - 2-е изд., перераб. и доп. - М. : Горячая линия-Телеком, 2007. - 336 с. : ил., табл. - (Учебник. Специальность для техникумов). - Библиогр.: с. 334.

Лабораторная работа № 4

Содержание

Цель работы:

- Изучить систему команд микропроцессора (МП) семейства «Intel mcs-51».
- Изучить среду отладки программ для системы команд МП «edsim51».

1. Введение

При разработке устройств большое внимание уделяется подсистемам ввода-вывода информации. Такие подсистемы позволяют воспринимать воздействия от окружающей среды и могут сами создавать такие воздействия. Для контроля и управления микропроцессорными устройствами существует интерфейс пользователя. Ввод информации в устройство, как правило, осуществляется переключателями, кнопочным блоком, аналого-цифровым преобразователем и т.п. Устройства вывода информации позволяют отображать состояние микропроцессорного устройства. Примером таких устройств могу служить светодиоды, семисегментные индикаторы, графические дисплеи, цифро-аналоговые преобразователи и т.п. Программы, обслуживающие работу интерфейсной части системы являются неотъемлемой частью любой микропроцессорной системы. Оттого, как будет реализована эта часть программного продукта, зависят функциональные и эргономические характеристики устройства. В данной лабораторной работе будут рассмотрены принципы работы с клавишным модулем и семисегментным индикатором. Благодаря их простоте использования и большим функциональным возможностям эти интерфейсные модули очень популярны в микропроцессорной технике.

2. Семисегментный индикатор

На рисунке 1 представлена принципиальная схема подключения семисегментного индикатора к микропроцессору.

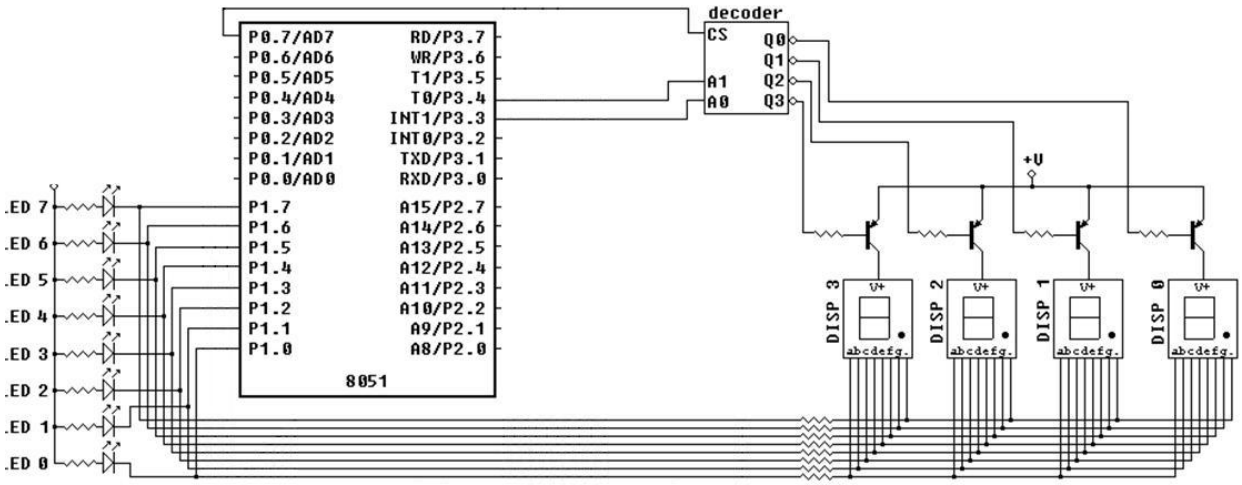


Рисунок 1 – Схема подключения семисегментного дисплея к микропроцессору

Как видно из рисунка 1, катоды индикаторов соединяются вместе и подключены к порту P1 микропроцессора. Такая схема включения называется динамической индикацией. Для того, чтобы индикатор засветил сегменты, надо, чтобы на аноде было положительное напряжение, а на катодах были «0». Выбор индикатора, на котором будет отображаться информация, определяется состоянием дешифратора и отображено в таблице 1, где «x» - любое значение («0» или «1»)

Таблица 1

CS	A0	A1	/Q0	/Q1	/Q2	/Q3
1	0	0	0	1	1	1
1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	1	1	1	1	0
0	x	x	1	1	1	1

Сигнал активации этого дешифратора (CS – Chip Select) подключен к выводу P0.7. Адресная линия этого дешифратора подключена к порту P3 микропроцессора. То есть, меняя значение на выводах P3.3, P3.4, можно менять номер индикатора, на котором будет отображаться информация.

Таким образом, если в порт P1, записать «0» во всех разрядах, P0.7 = «1», а P3.3 = P3.4 = «0», то в крайнем правом индикаторе зажгутся все сегменты.

Для того, чтобы сформировать код символа, для его отображения на индикаторе, нужно определить, какие сегменты индикатора должны зажечься. Эти сегменты обозначены «a», «b», «c», … «dp» (decimal point – десятичная точка), как показано на рисунке 2.

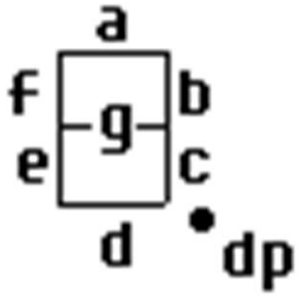


Рисунок 2 – Обозначение сегментов индикатора

Таким образом, чтобы отобразить символ «1», надо зажечь сегменты «b» и«c», а для того, чтобы отобразить символ «7», то зажечь еще надо будет и сегмент «a». Соответственно, символ «8.» отображается свечением всех сегментов. Если взглянуть на схему подключения дисплея к микропроцессору, которая представлена на рисунке 4, то можно заметить, что сегмент «a» подключен к выводу P1.0, сегмент «b» к P1.1, «dp» к P1.7. Если представить схему в виде таблицы сегментов и выводов порта, то можно сформировать информацию, так, как она представлена в Таблице 2.

Таблица 2

Сегмент	dp	g	f	e	d	c	b	a
вывод	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

Как было сказано выше, чтобы засветить сегмент, необходимо подать сигнал логического «0» на определенный катод индикатора. Таким образом, например чтобы высветить код символа «1», в порт надо записать бинарное число «11111001B»(символ «B» Binary – означает, что число записано в двоичной системе счисления), для того, чтобы отобразить символ цифры «7», записать надо «11111000B», для «8.» - соответственно «00000000B».

На рисунке 3 представлена блока схема алгоритма программы, которая последовательно выводит символы «1», «2» «3», «4» на разные индикаторы DISP0-3. Как видно из блок-схемы, алгоритм можно разделить на четыре одинаковые части, которые предназначены для работы с определенным индикатором.

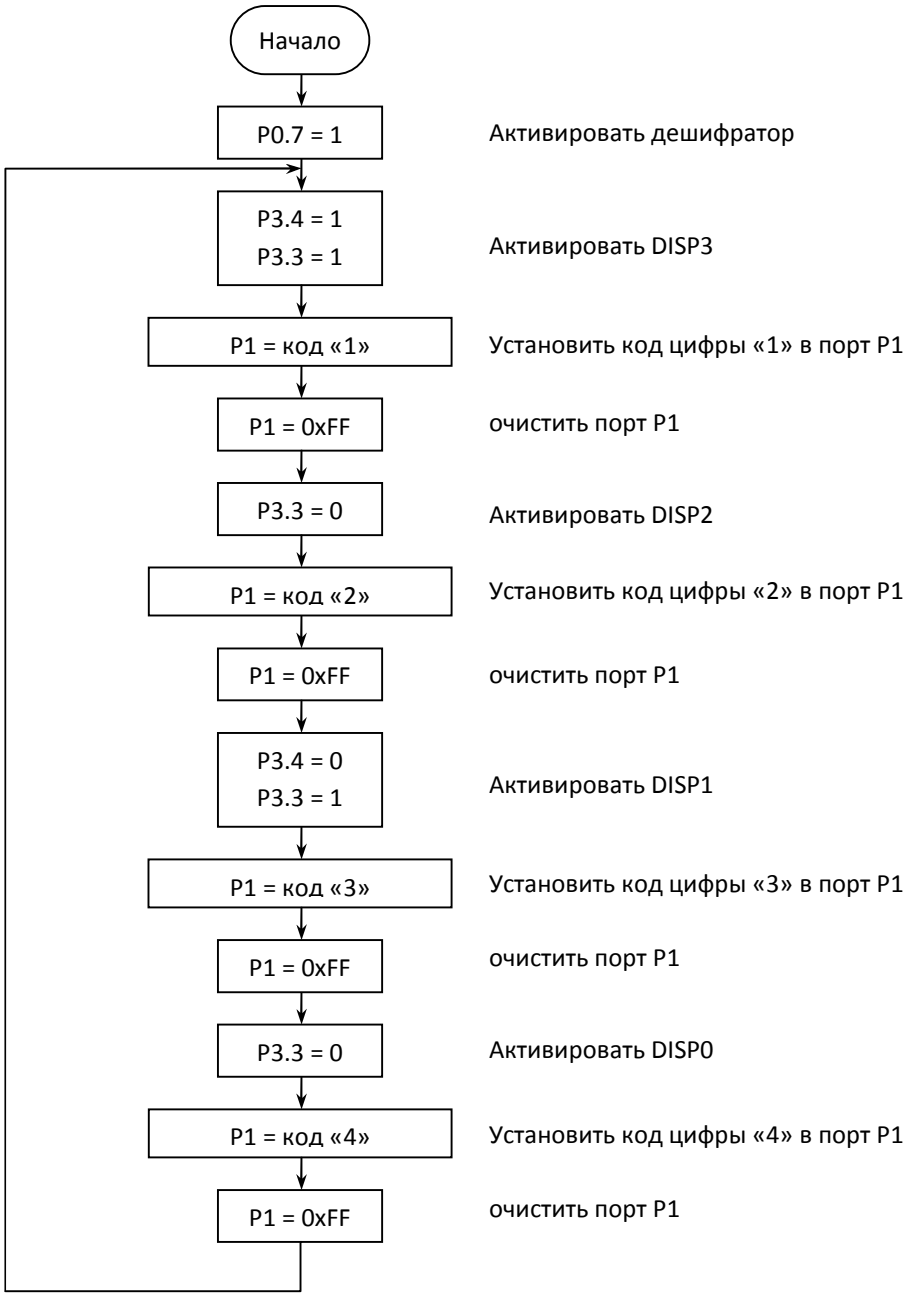


Рисунок 3 – Блок схема алгоритма вывода чисел «1», «2», «3», «4» на семисегментный дисплей

Пример программы, которая последовательно отображает числа от 1 до 4 на всех индикаторах, находится в папке «Examples», называется «SevenSegDisplays.asm» и приведена ниже:

```
          ; программа вывода символов 7-сегментный на дисплей
      SETB P0.7                    ; активировать дешифратор
start:
      SETB P3.3                    ; |
      SETB P3.4                    ; | активировать DISP3
      MOV P1, #11111001B           ; записать в порт код цифры «1»
      MOV P1, #0FFH                ; очистить порт
      CLR P3.3                     ; | активировать DISP2
      MOV P1, #10100100B           ; записать в порт код цифры «2»
      MOV P1, #0FFH                ; очистить порт
      CLR P3.4                     ; |
      SETB P3.3                    ; | активировать DISP1
      MOV P1, #10110000B           ; записать в порт код цифры «3»
      MOV P1, #0FFh                ; очистить порт
      CLR P3.3                     ; | активировать DISP0
      MOV P1, #10011001B           ; записать в порт код цифры «4»
      MOV P1, #0FFH                ; очистить порт
      JMP start                    ; перейти на начало программы
```

3. Клавишный модуль

На рисунке 4 показана электрическая принципиальная схема подключения клавишного модуля к микроконтроллеру. Если рассмотреть схему самого клавишного модуля, то его можно представить как сетку проводников, образующих строки и столбцы, на пересечении которых расположены кнопки, как показано на рисунке 2 . При нажатии на одну из кнопок, образуется замыкание определенной строки и столбца. Т.е. зная номера строки и столбца, можно определить, нажата кнопка с такими координатами или нет.

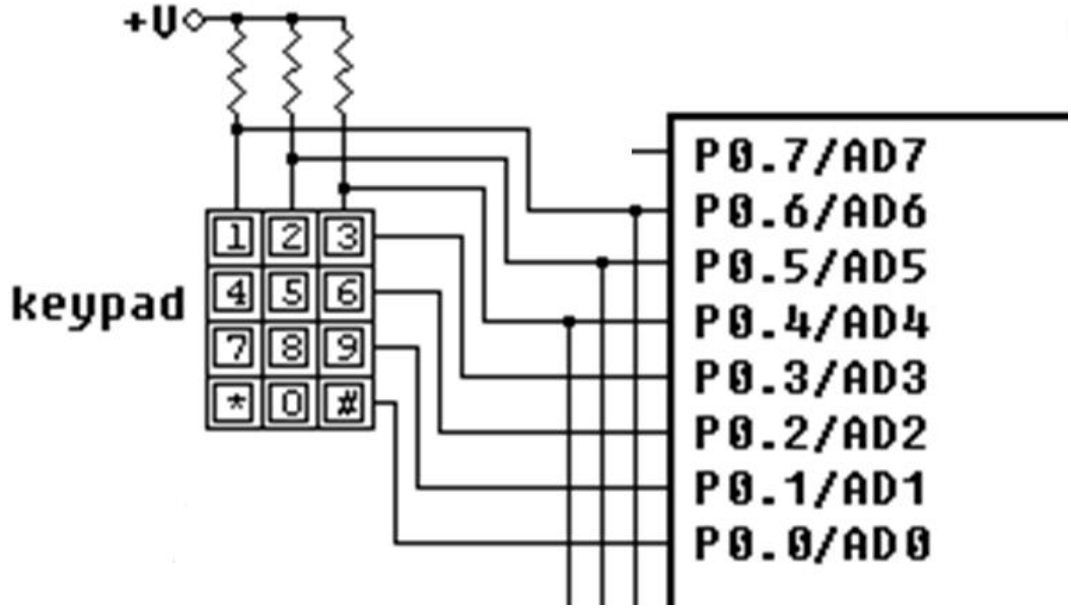


Рисунок 4 – Фрагмент принципиальной схемы отражающий подключение клавишного модуля.

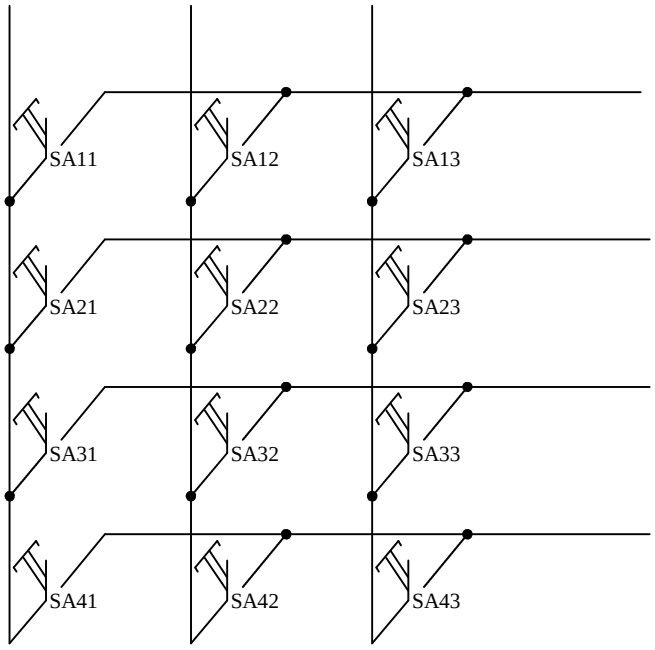


Рисунок 5 – Схемное представление клавишного модуля

Если посмотреть на рисунок 4, то можно заметить, что к столбцам клавишного модуля подключены подтягивающие резисторы. При помощи этих резисторов положительное напряжение поступает на вход микропроцессора. Таким образом, логический уровень «1» будет поступать на входы порта P0. Чтобы «0» поступил на порт P0.4-P0.6 необходимо сформировать логический уровень «0» на одной из строк клавишного модуля, подключенного к порту P0.0-P0.3 и нажать кнопку. Таким образом, если в определенный момент установить низкий логический уровень, например на выводе порта P0.0 и прочитать значение на входах P0.4-P0.6 (рис.4), то можно определить, нажаты ли кнопки SA41-SA43 (рис. 5). Если последовательно переключать низкий уровень на различных строках клавишного блока и читать значения со столбцов, то можно определить состояние всех кнопок в клавишном блоке. На рисунке 3 приведена блок-схема алгоритма сканирования клавишного модуля. Если посмотреть на этот алгоритм, то можно определить повторяющиеся участки. Эти участки определены по обработке строк клавишного модуля.

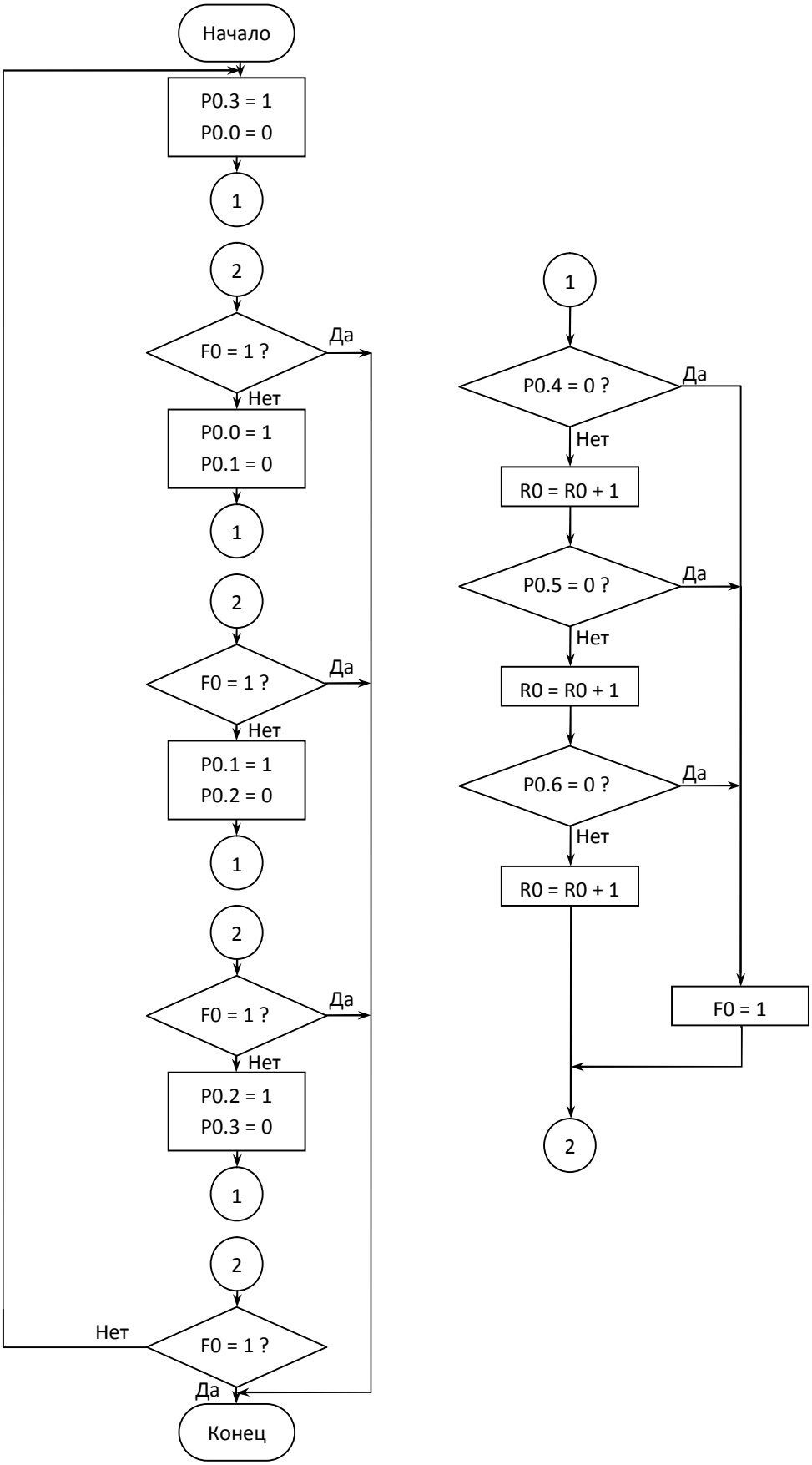


Рисунок 3 – Блок схема алгоритма сканирования клавишного блока

Ниже приведен пример кода, который определяет код нажатой клавиши и заносит его значение в регистр R0. Этот пример находится в папке «Examples» и называется «ScanKeypad.asm».

```
start:
```

```
    MOV R0, #0          ; очистить регистр R0
                            ; сканировать строку 0
    SETB P0.3           ; строка 3 = «1»
    CLR P0.0            ; строка 0 =  «0»
    CALL colScan        ; вызов подпрограммы сканирования столбцов
    JB F0, finish       ; | Если флаг F0 = «1», перейти на конец программы
                        ; | (так как сканированный код уже находится в
                        ; | регистре R0)
                            ; сканировать строку 1
    SETB P0.0           ; строка 0 = «1»
    CLR P0.1            ; строка 1 = «0»
    CALL colScan        ; вызов подпрограммы сканирования столбцов
    JB F0, finish       ; | Если флаг F0 = «1», перейти на конец программы
                        ; | (так как сканированный код уже находится в
                        ; | регистре R0)
                            ; сканировать строку 2
    SETB P0.1           ; строка 1 = «1»
    CLR P0.2            ; строка 2 = «0»
    CALL colScan        ; вызов подпрограммы сканирования столбцов
    JB F0, finish       ; | Если флаг F0 = «1», перейти на конец программы
                        ; | (так как сканированный код уже находится в
                        ; | регистре R0)
                            ; сканировать строку 3
    SETB P0.2           ; строка 2 = «1»
    CLR P0.3            ; строка 3 = «0»
    CALL colScan        ; вызов подпрограммы сканирования столбцов
    JB F0, finish       ; | Если флаг F0 = «1», перейти на конец программы
                        ; | (так как сканированный код уже находится в
                        ; | регистре R0)
    JMP start           ; | перейти на начало, чтобы сканировать строку 0
                        ; | поэтому вначале программы инициализируется
                        ; | строка 3 = «1»,так как при переходе в начало
                        ; | программы, строка 3 еще равна «0»
finish:
    JMP $               ; Останов программы, после определения кода клавиши

                            ; подпрограмма сканирования столбцов
colScan:
    JNB P0.4, gotKey    ; если столбец 0 = «0», перейти на подпрограмму
                        ; определения кода нажатой клавиши
    INC R0              ; иначе увеличить счетчик и проверить следующий
                        ; столбец
    JNB P0.5, gotKey    ; если столбец 1 = «0», перейти на подпрограмму
                        ; определения кода
    INC R0              ; иначе увеличить счетчик и проверить следующий
                        ;столбец
    JNB P0.6, gotKey    ; если столбец 2 = «0», перейти на подпрограмму
                        ; определения кода
    INC R0              ; иначе увеличить счетчик и проверить следующий
                        ; столбец
    RET                 ; возврат из подпрограммы в случае «клавиши не нажаты»
gotKey:
    SETB F0             ; Клавиша нажата, поэтому установить флаг F0 = «1»
    RET                 ; Возврат из подпрограммы в случае нажатой клавиши.
```

4. Задание к лабораторной работе

Составить алгоритм программы, которая сканирует код клавиши клавиатурного модуля и выводит на семисегментный дисплей его номер. При последующем нажатии клавиши, ее код отображается на самом правом индикаторе, а предыдущие значения сдвигаются влево, по аналогии набора цифры на калькуляторе. Написать, отладить и продемонстрировать программу на симуляторе «edsim51». В выводах необходимо привести минимальное достаточное время нажатия клавиши для определения ее кода.

5. Содержание отчета

В отчете необходимо привести следующие сведения:

- Цель лабораторной работы;
- Блок схема и описание алгоритма;
- Программный код на языке ассемблер;
- Выводы.

6. Контрольные вопросы

- Какой бинарный код нужно записать в порт P1, для того, чтобы отобразить символ цифры «5»?
- Как по-вашему, как следует организовать программу для того, чтобы определять две одновременно нажатые клавиши?
- Для чего нужна кнопка «AND Enabled/AND Disabled» на виртуальных панели устройств симулятора «edsim51». На что она влияет?
- Как следует организовать программу для того, чтобы сократить время определения нажатия кнопки клавишного блока?
- Что такое динамическая индикация? Сколько выводов микропроцессора потребовалось бы задействовать при подключении такого же количества 7-сегментных индикаторов, но по обычной схеме статической индикации?
- Для чего в программе сканирования клавишного блока используется флаг F0? Какую смысловую нагрузку он несет и в каком регистре расположен?

7. Список литературы

    1. Шарапов А. В. Проектирование микропроцессорных устройств : руководство к выполнению курсовых проектов (в том числе ГПО) для студентов специальности "Промышленная электроника" / А. В. Шарапов ; Федеральное агентство по образованию, Томский государственный университет систем управления и радиоэлектроники, Кафедра промышленной электроники. - Томск : ТУСУР, 2009. - 74 с. : ил. - Библиогр.: с. 74
    2. Калабеков Б. А. Цифровые устройства и микропроцессорные системы : Учебник для средних специальных учебных заведений связи / Б. А. Калабеков. - 2-е изд., перераб. и доп. - М. : Горячая линия-Телеком, 2007. - 336 с. : ил., табл. - (Учебник. Специальность для техникумов). - Библиогр.: с. 334.

Лабораторная работа № 5

Содержание

Цель работы:

- Изучить систему команд микропроцессора (МП) семейства «Intel mcs-51».
- Изучить организацию прерываний МП 8051.

1. Введение

Аналого-цифровые преобразователи (АЦП) являются незаменимыми устройствами в работе микропроцессорной техники с аналоговой электроникой. Благодаря АЦП, аналоговый сигнал преобразуется в цифровой код и преобразование это можно назвать «сигнал – код».

В микропроцессорных системах, в которых производятся такие преобразования, очень важным аспектом является время получения кода и его обработка. Период времени, определяющий момент поступления запроса на оцифровку аналогового сигнала до момента, когда оцифрованный код отобразится на выходе АЦП, называется временем дискретизации (квантования). В системах реального времени, при работе с сигналами, форма которых меняется, очень важно производить оцифровку сигналов и обработку кодов вовремя, так как, любая задержка в цепи «оцифровка-обработка» создает погрешность, которую сложно компенсировать. Поэтому очень важной характеристикой АЦП является его время дискретизации, которое определяет, за какое время данные гарантированно поступят на выход преобразователя с момента получения запроса на оцифровку. Чем меньше эта величина, тем лучше. Второй характеристикой АЦП является его разрешающая способность. Эта характеристика отражает разрядность цифрового кода, получаемого на выходе АЦП, и определяет, насколько точно оцифрованный код соответствует аналоговому сигналу. Эти характеристики отражают качество преобразователя и не подлежат программной компенсации.

Однако, если рассматривать микропроцессорную систему в целом, то можно определить еще одну характеристику, которая влияет на погрешность всей системы и если ее не минимизировать, то может возникнуть ситуация, когда дорогое аппаратное обеспечение не позволяет достичь хороших результатов из-за обеспечения программного. Речь идет о времени обработки запроса оцифрованного

сигнала. Именно для того, как сократить это время и минимизировать погрешность при обработке сигнала, предназначена данная лабораторная работа.

2. Взаимодействие с АЦП

На рисунке 1 показана принципиальная схема соединения АЦП и микропроцессора в среде моделирования «edsim51».

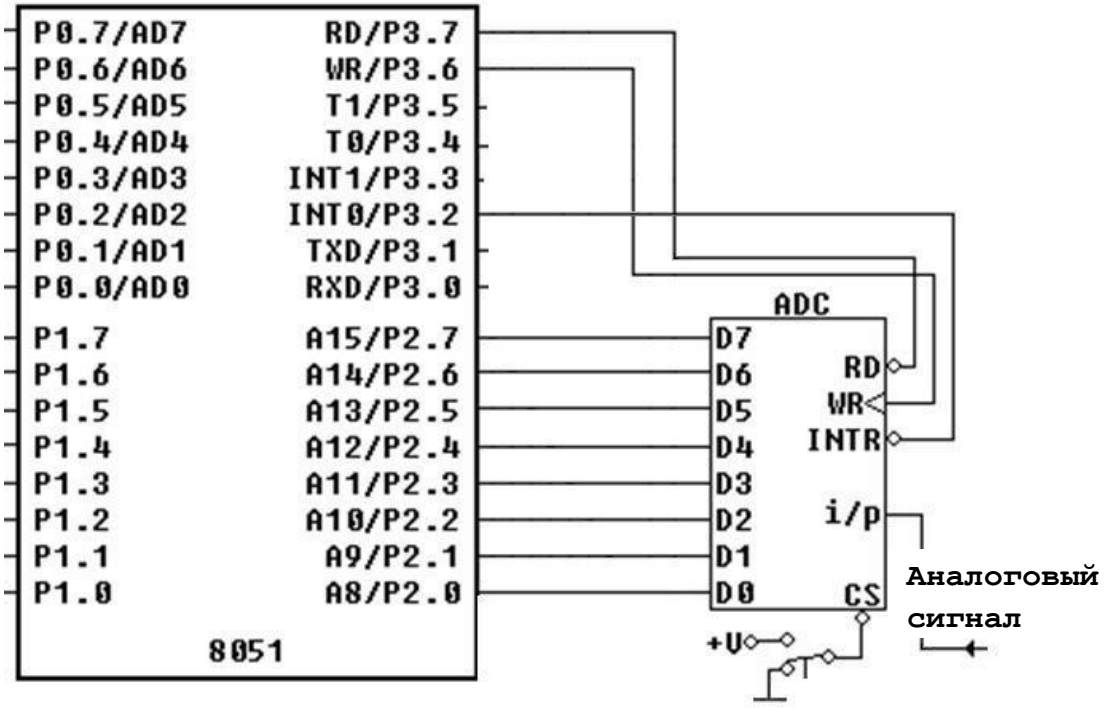


Рисунок 1 – Принципиальная схема соединения АЦП и микропроцессора в среде моделирования «edsim51»

Как видно из рисунка, АЦП (ADC – Analogue to Digital Converter) имеет следующие выводы:

/CS (Chip Select) – Вход выборки схемы, который активирует работу преобразователя при значении логического «0». При значении «1» схема игнорирует все внешние воздействия и не генерирует никаких сигналов.

i/p – Аналоговый вход преобразователя. Аналоговый сигнал, который подлежит дискретизации, подается на этот вход.

/WR – Сигнальный вход, запрос на оцифровку. Фронт импульса, при переходе из «0» в «1» производит запуск процесса дискретизации.

/INTR – Сигнальный выход, сообщающий микропроцессору, что процесс дискретизации завершен и требуется его участие для того, чтобы принять данные и подтвердить их получение.

/RD – Сигнальный вход, определяющий поведение линии данных АЦП. При низком логическом значении («0»), линия данных активна и способна передать код в микропроцессор. При переходе из низкого уровня в высокий (фронт импульса «0» -> «1»), подтверждается прием данных, запрос на обслуживание (/INTR) сбрасывается и линия данных переходит в пассивное состояние.
D0-D7 – выход, по которому в микропроцессор передается оцифрованный код сигнала.

Таким образом, исходя из описания интерфейса АЦП, можно составить алгоритм, изображенный на рисунке 2, при помощи которого можно произвести оцифровку аналогового сигнала.

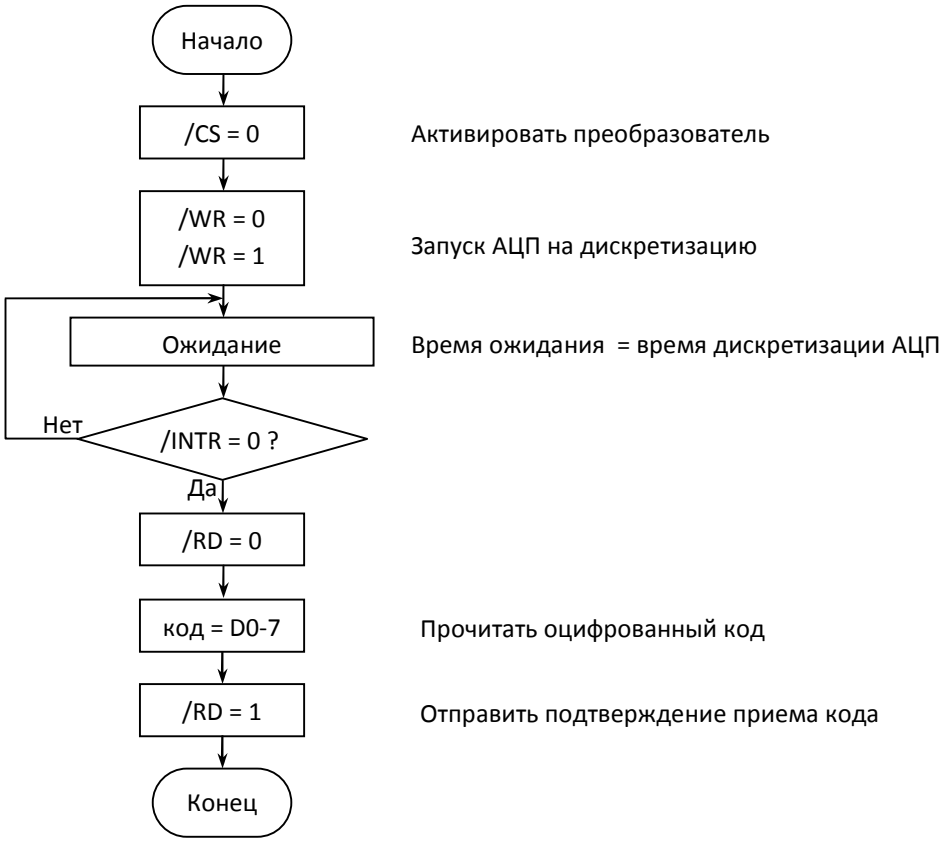


Рисунок 2 – Блок схема алгоритма работы с АЦП

3. Организация прерываний

Как было сказано выше, время дискретизации, разрешающая способность (разрядность АЦП) и время обработки кода определяет погрешность измерений микропроцессорной системы. Если время квантования сигнала определено и описано разработчиком АЦП, то его можно компенсировать. Время обработки зависит от производительности микропроцессора, который участвует в этой системе. Если организовать программное обслуживание АЦП в теле

основной программы (основного цикла) и вычислить период цикла, то можно увидеть, что это время зависит о количества команд, исполняемых в общем цикле. Так как микропроцессорные системы отрабатывают взаимодействие со многими устройствами (клавишный модуль, дисплей), то оценить задержку на обработку, а тем более минимизировать ее, становится затруднительной задачей для программиста. Соответственно, когда возникает вопрос оценки времени на обработку, вычисленная погрешность оказывается больше погрешности самого АЦП. Для того, чтобы минимизировать время обработки внешних устройств и сократить погрешности в системах реального времени, используется механизм прерываний и их обработка.

Основной особенностью в организации прерываний является тот момент, что когда возникает прерывание, исполнение программы, которое выполняется в основном потоке, прерывается и ресурсы процессора передаются программе обработчика прерывания. Как правило, это возникает потому, что в механизме обработки прерываний присутствует блок, который определяет приоритет выполнения. В ситуациях, когда два и более потока должны исполняться процессорным ядром, решение, какому потоку передать управление, определяет блок приоритетов или приоритетный арбитр. В составе этого блока присутствует регистр, в котором хранится текущие значения исполняемых прерываний и значения их приоритетов. Когда два потока и более запрашивают процессорные ресурсы, управление передается тому потоку, приоритет выполнения которого выше. Как правило, приоритет основного потока (т.н. фоновый режим) имеет наименьший приоритет. Это сделано с той целью, чтобы обрабатывать прерывания.

Запросы, которые могут прервать выполнение программы микропроцессора 8051, очень ограничены. Прерывания, которые обрабатываются этим микропроцессором, делятся на две категории: прерывания от таймеров и прерывания от внешних устройств.

Таймер это устройство, которое физически расположено в теле микропроцессора и представляет собой счетчик с сигнальным механизмом. После того, как счетчик был проинициализирован значением и был произведен старт, значение в счетчике начинает

изменяться линейно со временем (увеличиваться или уменьшаться). Когда значение счетчика достигает определенной границы, сигнальное устройство генерирует импульс, по которому производится действие. Вместе с этим, счетчик инициализируется прежним значением и последовательность действий повторяется. В качестве основного достоинства применения таймера можно отметить то, что в такой организации можно получить последовательность импульсов, с определенным периодом, который не связан с исполнением основной программы или с обработкой прерываний (таймер изменяет значение в счетчике независимо от работы ядра процессора).

Прерывания от внешних устройств организованы при помощи детектора, который программируется на возникновение определенного события (фронт сигнала, логический уровень). По конструкции блок обработки внешних прерываний организован проще, чем блок организации таймера и обработки его прерываний.

Упрощенная схема прерываний показана на рисунке 3. Адреса по векторам 0003H, 0013H соответствуют внешним прерываниям. Внешние прерывания INT0 и INT1 могут быть вызваны либо уровнем, либо переходом сигнала из «1» в «0» на входах микропроцессора, в зависимости от значений управляющих бит «IT0» и «IT1» в регистре «TCON». От внешних прерываний устанавливаются флаги «IE0» и «IE1» в регистре «TCON», которые инициируют вызов соответствующей программы обслуживания прерывания. Сброс этих флагов выполняется аппаратно только в том случае, если прерывание было вызвано по переходу (фронту) сигнала. Если же прерывание вызвано уровнем входного сигнала, то сбросом флага «I» должна управлять соответствующая подпрограмма обслуживания прерывания путем воздействия на источник прерывания с целью снятия им запроса. Флаги запросов прерывания от таймеров «TF0» и «TF1» сбрасываются автоматически при передаче управления подпрограмме обслуживания. Флаги запросов прерывания «RI» и «TI» устанавливаются блоком управления приемопередатчика аппаратно, но сбрасываться должны программным путем.

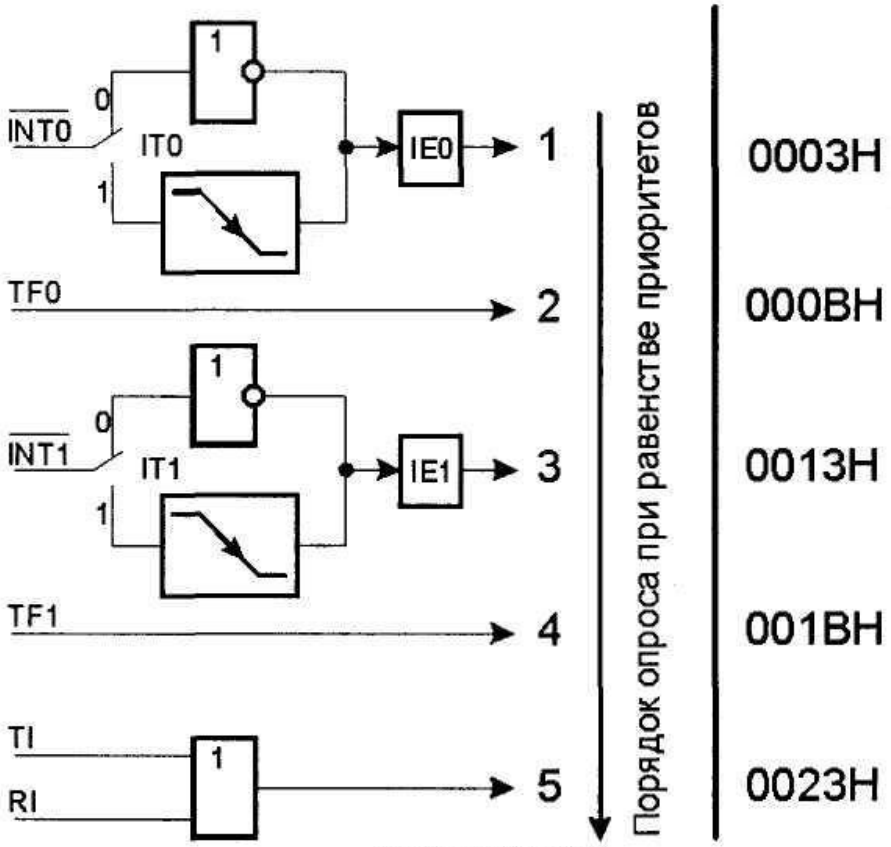


Рисунок 3 - Схема и распределение векторных областей микропроцессора Intel MCS-51

Прерывания могут быть вызваны или отменены программой, так как все названные флаги программно доступны и могут быть установлены/ сброшены программой с тем же результатом, как если бы они были установлены/сброшены аппаратными средствами.

В блоке регистров специальных функций есть два регистра, предназначенных для управления режимом прерываний «IE» и уровнями приоритета «IP». Возможность программной установки/сброса любого управляющего бита в этих двух регистрах делает систему прерываний 8051 исключительно гибкой.

В более сложных модификациях микроконтроллеров семейства MCS-51 количество периферийных устройств увеличено, что приводит к необходимости использовать один вектор прерывания для нескольких устройств (разделение подпрограмм обслуживания прерываний в этом случае необходимо реализовать программно), либо добавить еще два регистра - режима (маски) и приоритета прерываний. Регистр масок прерываний определяет, какое прерывание будет обрабатываться, а какое нет, как показано в таблице 1.

Таблица 1

Символ	Позиция	Имя и назначение
EA	IE.7	Снятие блокировки прерывания. Сбрасывается, программно для запрета всех прерываний независимо от состояний IE.4 - IE.0
	IE.6	Не используется
	IE.5	Не используется
ES	IE.4	Бит разрешения прерывания, от **приемопередатчика** Установка/сброс программой для разрешения/запрета прерываний от флагов TI или RI .
ET1	IE.3	Бит разрешения прерывания от **таймера 1**. Установка/сброс программой для разрешения/запрета прерываний от таймера 1
EX1	IE.2	Бит разрешения **внешнего прерывания 1**. Установка/сброс программой для разрешения/запрета прерывания 1
ETO	IE.1	Бит разрешения прерывания от **таймера 0**. Установка/сброс программой для разрешения/запрета прерываний от таймера 0 .
EXO	IE.0	Бит разрешения **внешнего прерывания 0**. Установка/сброс программой для разрешения/запрета прерывания 0

Описание регистра приоритетов представлено в таблице 2.

Таблица 2

Символ	Позиция	Имя и назначение
-	IP.7-IP.5	Не используется
PS	IP.4	Бит приоритета **приемопередатчика**. Установка/сброс программой для присваивания прерыванию от приемопередатчика высшего/низшего приоритета
PT1	IP.3	Бит приоритета **таймера 1**. Установка/сброс программой для присваивания прерыванию от таймера 1 высшего/низшего приоритета
PX1	IP.2	Бит приоритета внешнего прерывания 1. Установка/сброс программой для присваивания высшего/низшего приоритета внешнему прерыванию **INT1**
PT0	IP.1	Бит приоритета **таймера 0**. Установка/сброс программой для присваивания прерыванию от таймера 0 высшего/низшего приоритета
PX0	IP.0	Бит приоритета внешнего прерывания 0. Установка/сброс программой для присваивания высшего/низшего приоритета внешнему прерыванию **INTO**

4. Выполнение подпрограммы прерывания

Система прерываний формирует аппаратный вызов (LCALL) соответствующей подпрограммы обслуживания, если она не заблокирована одним из следующих условий:

- в данный момент обслуживается запрос прерывания равного или высшего уровня приоритета;
- текущий машинный цикл — не последний в цикле выполняемой команды;
- выполняется команда RETI или любая команда, связанная с обращением к регистрам IE или IP.

Если флаг прерывания был установлен, но по одному из указанных выше условий не получил обслуживания и к моменту окончания блокировки уже сброшен, то запрос прерывания теряется и нигде не

запоминается. По аппаратно сформированному коду LCALL, система прерывания помещает в стек только содержимое счетчика команд (PC) и загружает в него адрес вектора соответствующей подпрограммы обслуживания. По адресу вектора должна быть расположена команда безусловной передачи управления (JMP) к начальному адресу подпрограммы обслуживания прерывания. В случае необходимости она должна начинаться командами записи в стек (PUSH) слова состояния программы (PSW), аккумулятора, расширителя, указателя данных и т.д. и должна заканчиваться командами восстановления из стека (POP). Подпрограммы обслуживания прерывания должны завершаться командой RETI, по которой в счетчик команд перезагружается из стека сохраненный адрес возврата в основную программу. Команда RET также возвращает управление прерванной основной программе, но при этом она не снимает блокировку прерываний, что приводит к необходимости иметь программный механизм анализа и окончания процедуры обслуживания данного прерывания.

5. Организация работы АЦП по прерыванию

На рисунке 4 представлена блок – схема алгоритма работы программы, которая читает код, получаемый из АЦП и передает это значение на вход ЦАП.

Рисунок 4 – Блок схема алгоритма программы

Синхронизация программы выполнена при помощи таймера, период которого задается программно на этапе инициализации.
Пример программы называется «ADCtoDAC.asm», находится в папке «examples» и приведен ниже:

```
                        ; инициализация векторов прерываний
ORG 0                   ; вектор прерывания при сигнале аппаратного сброса
      JMP main          ; переход к выполнению основной программы

ORG 3                   ; вектор внешнего прерывания INT0
      JMP ext0ISR       ; переход на обработчик внешнего прерывания

ORG 0BH                 ; вектор прерывания от устройства Timer0
      JMP timer0ISR     ; переход на обработчик прерывания от таймера

ORG 30H                 ; организация адресного пространства основной
                        ; программы
main:
      SETB IT0          ; установить детектор внешнего прерывания на фронт
      SETB EX0          ; разрешить прерывания от внешнего устройства 0
      CLR P0.7          ; активировать работу ЦАП
      MOV TMOD, #2      ; установить Таймер 0 как 8-битный, автозагружаемый,
                        ; периодический счетчик

      MOV TH0, #-20     ; | установить значение в старший байт счетчика для
                        ; | последующей автозагрузки (-20 = 256-20 = 236)
      MOV TL0, #-20     ; | установить значение в младший байт счетчика
                        ; | Таймера 0, использования при подсчете первого
                        ; | периода.
      SETB TR0          ; Запустить таймер 0
      SETB ET0          ; Разрешить прерывания от Таймера 0
      SETB EA           ; Разрешить прерывания, установить глобальный бит
                        ; разрешения прерываний
      JMP $             ; Останов программы (ловушка)

; Конец основной программы

; обработчик прерывания от таймера: запуск АЦП на дискретизацию
timer0ISR:
      CLR P3.6          ; установить вывод WR = 0
      SETB P3.6         ; установить вывод WR = 1, при фронте импульса
                        ; «1» -> «0» производится запуск АЦП на дискретизацию
      RETI              ; возврат из обработчика

; обработчик прерывания от внешнего устройства – АЦП:
; активирует линиею данных АЦП,
; копирует оцифрованный код из порта P2 в порт P1,
;.сбрасывает запрос на прерывание от внешнего устройства - АЦП
ext0ISR:
      CLR P3.7          ; Разрешить передачу кода с АЦП по линии данных
      MOV P1, P2        ; Скопировать код из порта P2 в порт P1
      SETB P3.7         ; Сбросить запрос на прерывание и перевести линию
                        ; данных в пассивное состояние
      RETI              ; возврат из обработчика
```

6. Задание к лабораторной работе

Составить алгоритм программы «цифровой вольтметр», написать, отладить и продемонстрировать программу для симулятора «edsim51». Микропроцессор должен получать оцифрованный код с АЦП, преобразовывать его и отображать на семисегметном дисплее в формате входного напряжения. Т.е. если на входе АЦП 3.32 вольта, то на дисплее должно высвечиваться такое же значение. В заключении необходимо привести погрешность измерений и определить их источник.

7. Содержание отчета

В отчете необходимо привести следующие сведения:

- Цель лабораторной работы;
- Блок схема и описание алгоритма;
- Программный код на языке ассемблер;
- Выводы.

8. Контрольные вопросы

- Какой код будет на выходе АЦП, симулятора «edsim51», если на его входе 5 Вольт?
- Для чего нужны прерывания?
- Какое функциональное значение у таймера?
- Что такое время дискретизации АЦП?
- Чем отличаются команды RET и RETI?
- Что такое приоритет прерываний и как его можно изменить?
- Как в вашей программе можно изменить период дискретизации входного сигнала?

9. Список литературы

1. Калабеков Б. А. Цифровые устройства и микропроцессорные системы : Учебник для средних специальных учебных заведений связи / Б. А. Калабеков. - 2-е изд., перераб. и доп. - М. : Горячая линия-Телеком, 2007. - 336 с. : ил., табл. - (Учебник. Специальность для техникумов). - Библиогр.: с. 334.

Лабораторная работа № 6

Содержание

Цель работы:

- Изучить систему команд микропроцессора (МП) семейства «Intel mcs-51».
- Изучить организацию взаимодействий внешних устройств с микропроцессором.

8. Введение

Цифро-аналоговые преобразователи (ЦАП) предназначены для формирования аналогового сигнала из цифрового кода. Такое преобразование позволяет управлять аналоговыми электронными схемами при помощи микропроцессора. При помощи АЦП и ЦАП аналоговый сигнал может преобразоваться в цифровой код и обратно. Если же между этими устройствами использовать микропроцессор, то можно производить цифровую обработку сигнала, которая используется во многих цифровых устройствах и системах.

Генерация синтезированного сигнала так же широко используется в различных системах. Такие системы состоят лишь из микропроцессора, ЦАП и интерфейсной части. Главная задача состоит в том, чтобы форма сигнала как можно точнее соответствовала «идеализированной» форме.

Основным технических параметром ЦАП является (как и по аналогии с АЦП) время преобразования и шаг преобразования. Время преобразования определяет момент появления преобразованного аналогового напряжения на выходе, с момента поступления его кода на входе ЦАП. Т.е. это время, за которое произойдет преобразование цифрового кода в аналоговый сигнал. Шаг преобразования определяет точность преобразователя и зависит от разрядности линии данных, которая поддерживается ЦАП. Т.е., например, если разрядность ЦАП – 16 бит, а напряжение на выходе может изменяться в пределах 1 Вольта, то можно вычислить шаг преобразования. Так как в 16 битах можно отобразить 65536 различных значений, то разделив 1 Вольт на количество значений, можно сказать, насколько изменится выходное напряжение, при изменении кода в самом младшем бите. 1/65535 = 0.00001525 Вольта или 15.3 микровольта.

Размер минимального шага преобразования напрямую зависит от амплитуды выходного сигнала ЦАП, поэтому сравнить различные ЦАП по одним значениям минимального шага не возможно. В этом случае, лучше произвести сравнение этих устройств по разрядности их линий входных данных.

9. Организация работы с ЦАП

Принципиальная схема подключения ЦАП к микропроцессора в среде моделирования «edsim51» показана на рисунке 1.

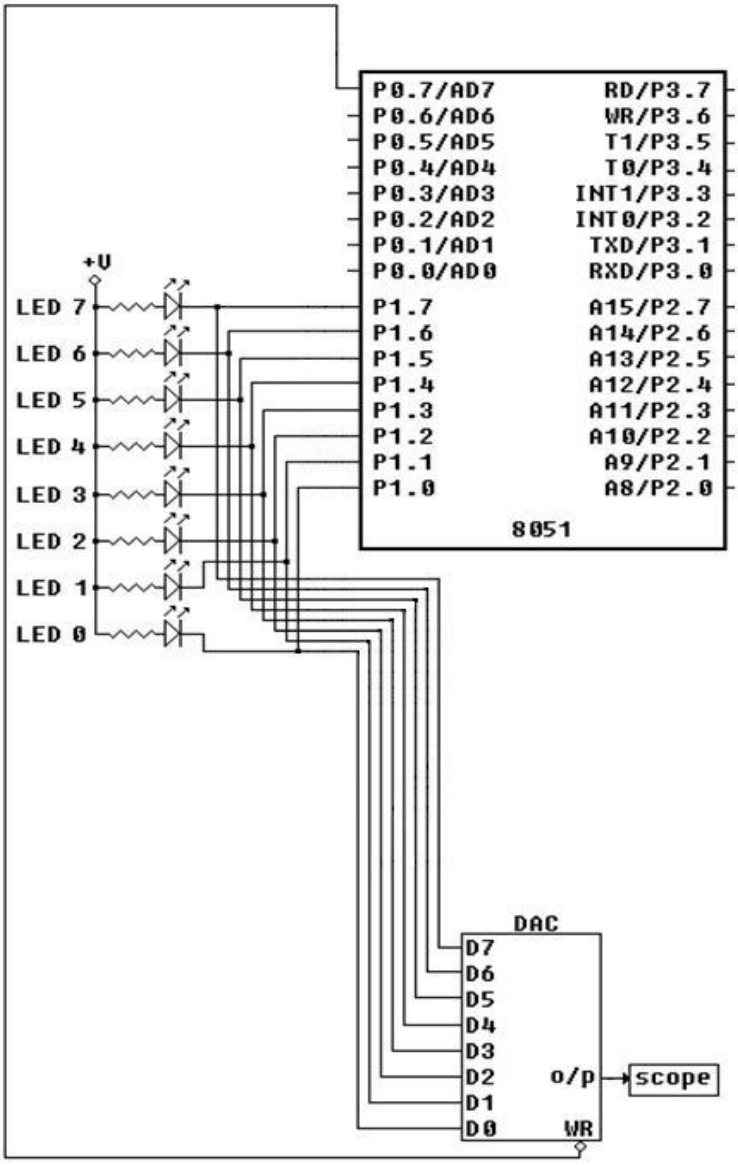


Рисунок 1 – Схема подключения ЦАП к микропроцессору в среде моделирования «edsim51»

ЦАП, показанный на рисунке 1, содержит следующие выводы:

D0-7 – линия входных данных цифрового кода;

o/p – выход аналогового сигнала, соединен в симулятором осциллографа, который отображает форму генерируемого сигнала;
/WR – сигнальный вход, определяющий запрос на преобразование. В случае, когда /WR = «0», цифровой код, представленный на входе D0-7, будет преобразован в аналоговый сигнал, который появится на выводе «o/p». Если /WR = «1» преобразование производиться не будет и на вывод «o/p» будет отключен.
Таким образом, для того, чтобы получить аналоговый сигнал из цифрового кода, достаточно установить запрос на преобразование /WR = «0» и цифровой код на входах D0-7.

10. Программирование ЦАП

На рисунке 2 представлена блок – схема алгоритма работы программы, которая генерирует пилообразный сигнал на выходе ЦАП.

Рисунок 2 – Блок схема алгоритма программы

Пример этой программы называется «DAC.asm», находится в папке «examples» и приведен ниже:

```
      CLR P0.7    ; разрешить работу ЦАП
loop:
      MOV P1, A   ; записать в порт значение, хранимое в аккумуляторе
      ADD A, #8   ; увеличить значение в аккумуляторе на 8
      JMP loop    ; перейти на начало программы
```

4. Задание к лабораторной работе

Составить алгоритм программы «программируемый генератор сигналов». Программа должна сканировать код клавиши (например клавиши «1», «2», «3») и при их нажатии должна переключаться в соответствующий режим генерации сигнала. Формы сигналов должны быть трех видов: прямоугольный, пилообразный, синусоидальный. Амплитуда сигнала должна изменяться в зависимости от входного напряжения, поступающего на вход АЦП. Написать, отладить и продемонстрировать программу для симулятора «edsim51». В выводах необходимо отразить значение периода и частоты полученных сигналов.

5. Содержание отчета

В отчете необходимо привести следующие сведения:

- Цель лабораторной работы;
- Блок схема и описание алгоритма;
- Программный код на языке ассемблер;
- Выводы.

6. Контрольные вопросы

- Что такое ЦАП и какую функцию он выполняет?
- Какие основные технические характеристики ЦАП?
- Какой уровень сигнала будет на выходе ЦАП, симулятора «edsim51», если на его входе будет код 0x7F?
- Как в вашей программе можно изменить период генерируемого сигнала?
- Оцените минимальный шаг преобразования для ЦАП симулятора «edsim51».

7. Список литературы

1. Шарапов, А. В. Основы микропроцессорной техники : учебное пособие / А. В. Шарапов ; Федеральное агентство по образованию, Томский государственный университет систем управления и радиоэлектроники. - Томск : ТУСУР, 2008. - 240 с.

Лабораторная работа № 7

Содержание

Цель работы:

- изучение архитектуры микроконтроллеров семейства Intel 8051, на основе синтезируемой модели.
- изучение среды моделирования,
- исследование проекта, запуск процесса симуляции и анализ полученных данных.

1. Введение

Однокристальные микропроцессоры (МП) семейства Intel 8051 были разработаны компанией Intel в 1980 году. Эти МП имеют Гарвардскую архитектуру и используются во встраиваемых системах. Первые версии таких микроконтроллеров были популярны с 80-х и до середины 90-х годов. Сегодня большое количество производителей изготавливают МП, совместимые с МП ранних версий семейства Intel 8051.

2. Архитектура исследуемой модели

Архитектура микроконтроллера имеет иерархическую структуру. Самый верхний уровень в иерархии представлен в файле A8051_exp.bde. Этот файл позволяет графически отобразить внутренние модули и связи между ними.

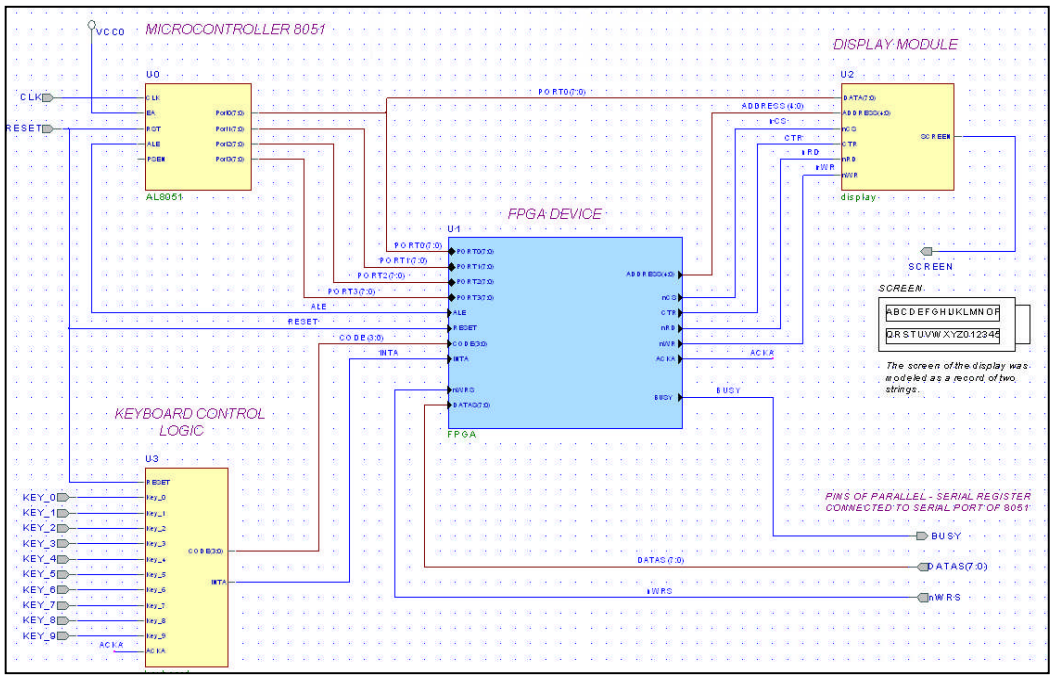
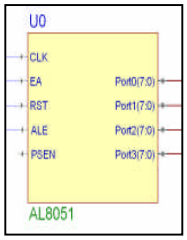


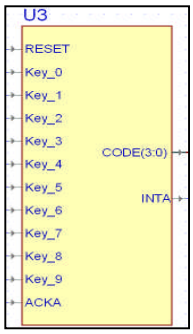
Рисунок 1 – Графическое представление синтезируемой модели

Как видно из рисунка 1, проект состоит из следующих модулей:

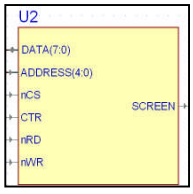
- Контроллер AL8051 (Microcontroller 8051) – модуль, который описывает поведение МП семейства 8051.



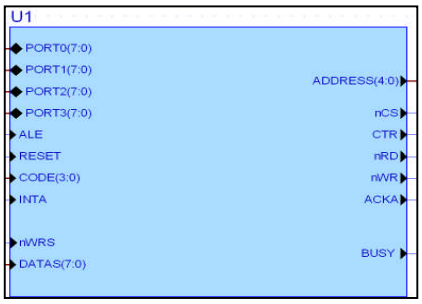
- Клавиатура (Keyboard control Logic) – модуль, который имитирует и контролирует логику работы клавиатуры.



- Дисплей (Display)- модуль предназначен для отображения символьной информации. По структурной организации этот модуль состоит из 32-х запоминающих элементов, в каждом из которых хранится символ и отображается на индикаторе.



- Матрица логических элементов (FPGA Device). Этот модуль организует взаимодействие между основными компонентами.



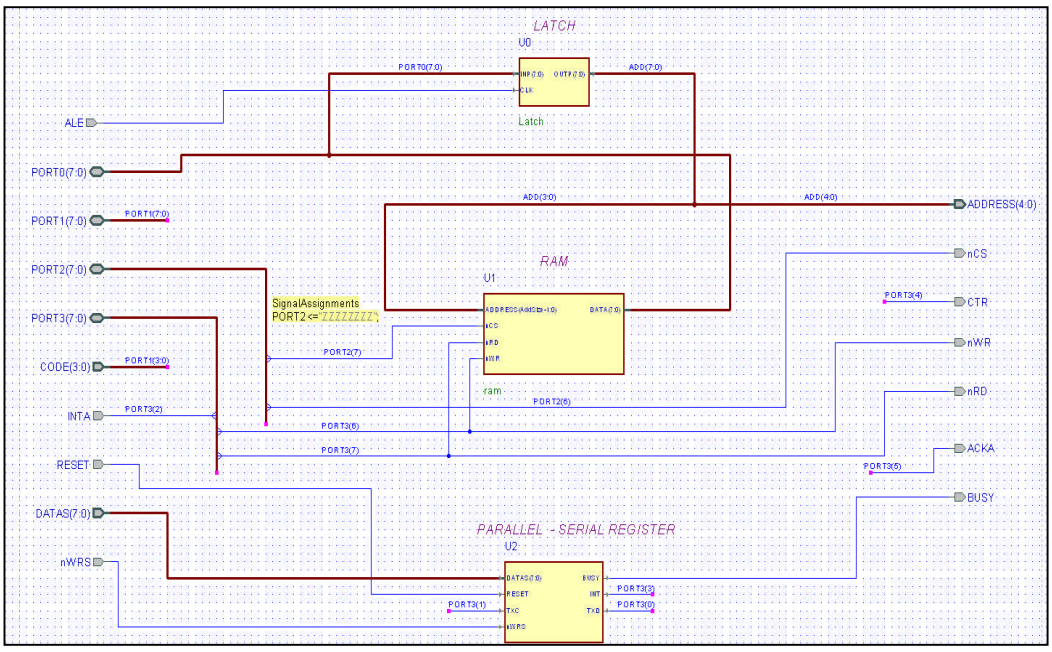
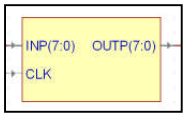


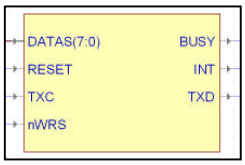
Рисунок 2- Структура матрицы логических элементов

Структура матрицы представлена на рисунке 2 и состоит из 3-х элементов:



- Latch – модуль предназначен для соединения внешних устройств с ядром микроконтроллера.

- RAM -модуль предназначен для работы микроконтроллера и состоит из 16-ти ячеек 8-ми разрядной памяти.



- Parallel-Serial Register. Этот модуль предназначен для преобразования способа передачи данных из параллельного в последовательный формат.

3. Рабочее поведение модели

Поведение модели зависит от программы, которая исполняется ядром микроконтроллера. Файл исходного кода программы PROGRAM.ASM находится в корневой директории проекта. В начале симуляции на дисплейный модуль отправляется сообщение «*** HELLO *** IP Core 8051», после этого сообщение изменяется: «PRESS ANY KEY IP Core 8051» Если установить логическое значение «1» на вход клавиатурного модуля, программа отобразит номер клавиши. Например, если установить «1» на вход «KEY_3» (нажата виртуальная клавиша «3»), то ядро микроконтроллера отправит сообщение «PRESS ANY KEY KEY NUMBER 3» Клавиатурный модуль генерирует прерывание в момент нажатия виртуальной кнопки, что позволяет подробно исследовать процесс обработки прерывания.

4. Изучение среды моделирования

Запуск среды моделирования «Active-HDL» можно произвести несколькими способами, например можно найти на рабочем столе иконку с однозначным названием, либо воспользоваться кнопкой «Пуск», программы и.т.д. После запуска можно наблюдать диалоговое окно, при помощи которого можно открыть проект, как представлено на рисунке 3.

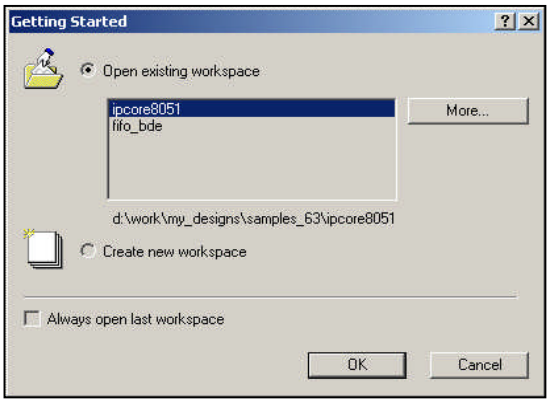


Рисунок 3 - Диалоговое окно выбора открываемого проекта

Проект «ipcore8051» можно открыть из окна проектов, нажав кнопку «ОК», либо выбрать интересующий проект из списка, нажав на кнопку «More…», как показано на рисунке 4.

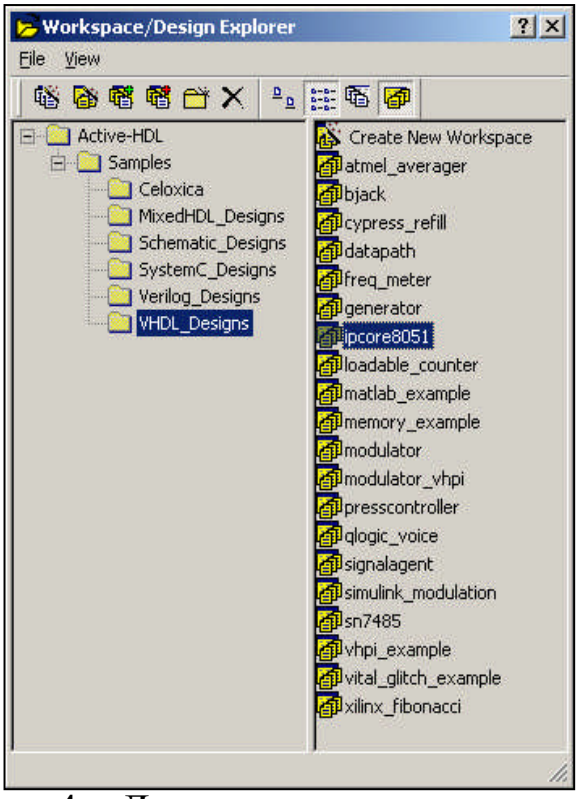


Рисунок 4 – Диалоговое окно проводника проектов

Двойным нажатием левой кнопки мышки открываем проект «ipcore8051». После чего появится окно среды разработки и моделирования «Active HDL», как показано на рисунке 5.

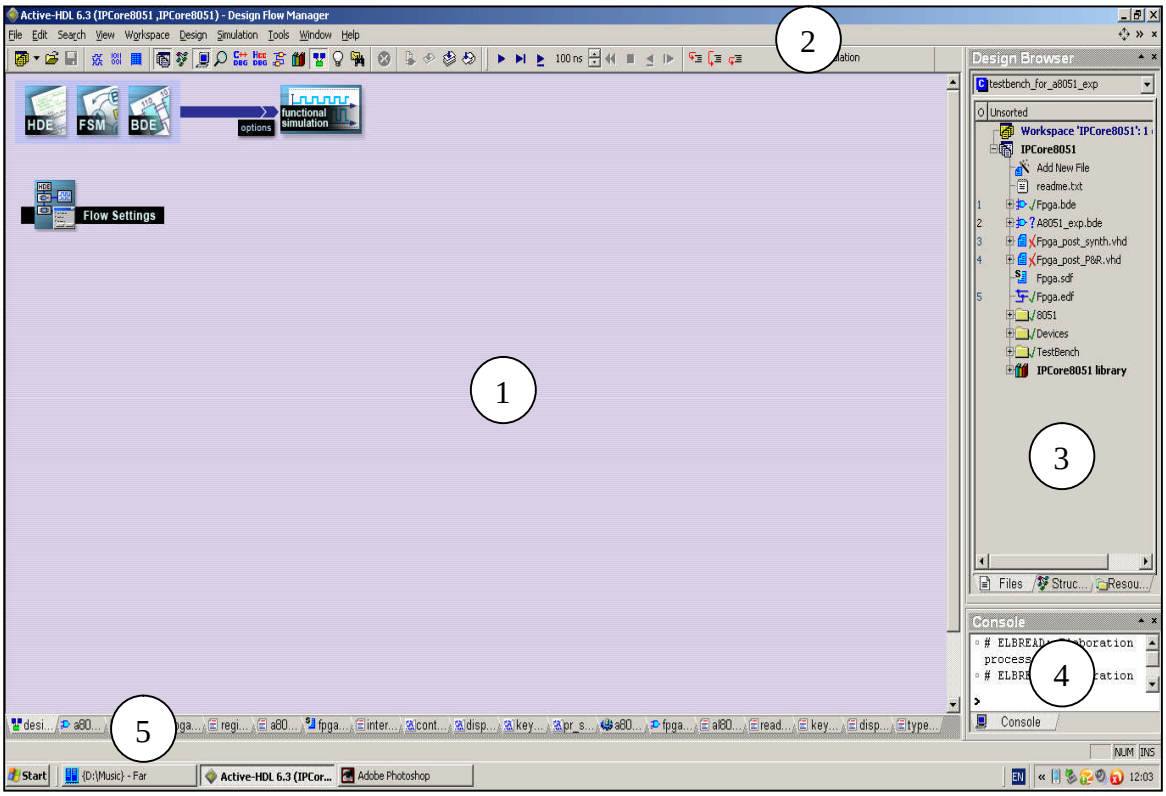


Рисунок 5 – Основное окно среды разработки «Active - HDL»

Как видно из рисунка 5, окно среды разработки состоит из 5 основных частей:

- Диалоговое окно, на котором отображается основная информация проекта (область 1). Отображение бывает как в графическом виде (рис.1, 2), так и в текстовом, как показано на рисунке 6.

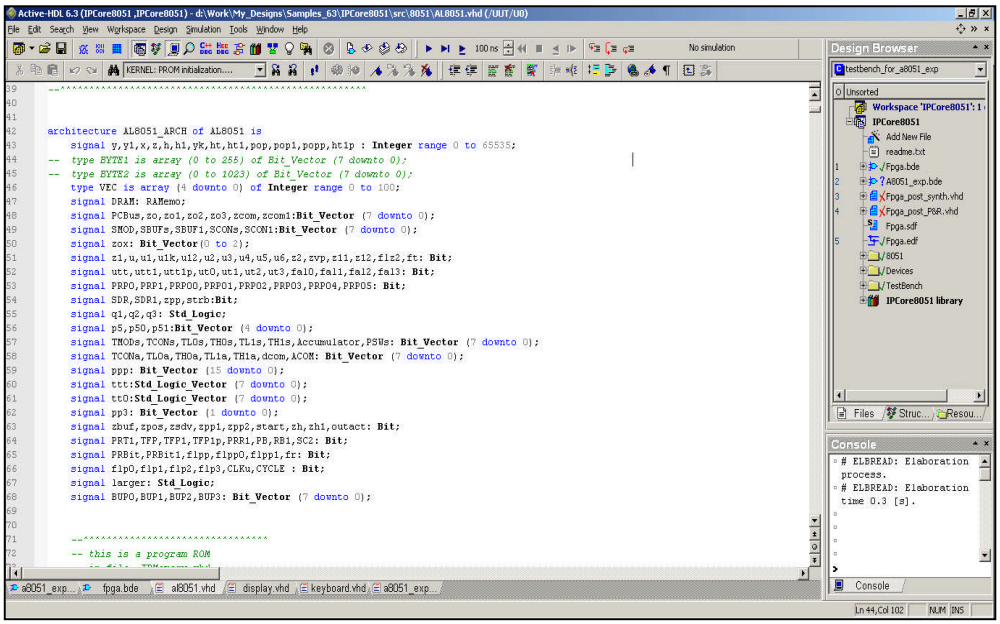
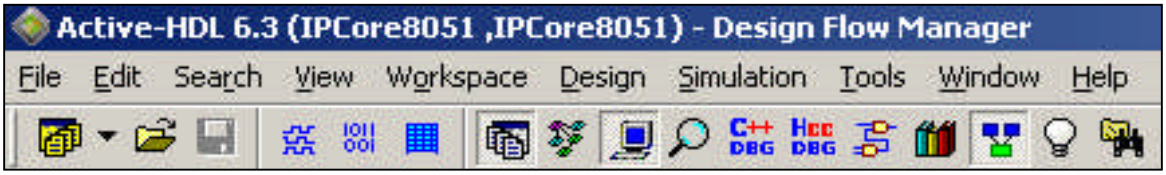


Рисунок 6 – Среда разработки и моделирования (текстовое представление проекта)

- Область панели инструментов (область 2). Эта область предназначена для управления проектными данными и поведением среды моделирования. Пример представления приведен на рисунке 7.

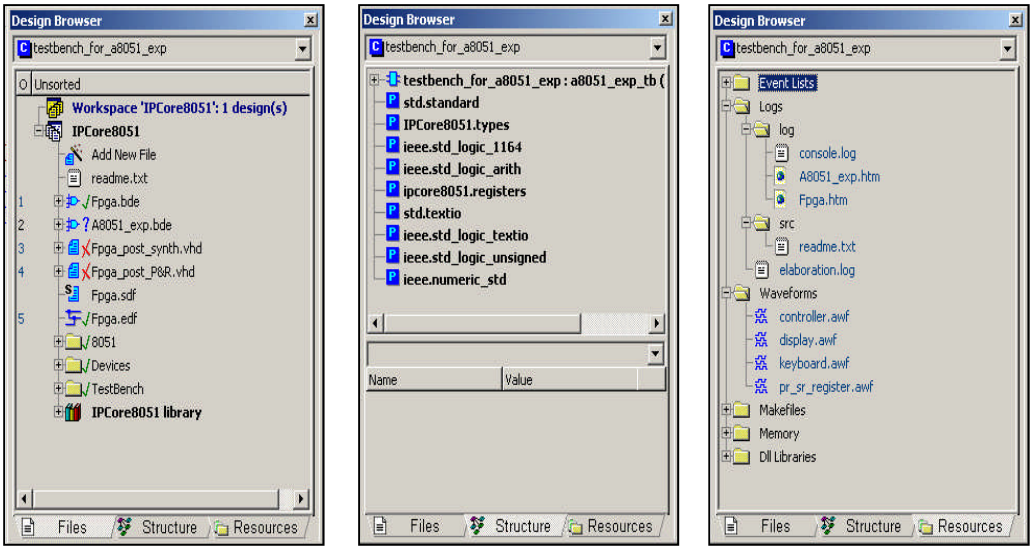


а)

б)

Рисунок 7 – Область панелей , где а - Панель инструментов и меню, б - Панель управления процессом симуляции

- Окно проекта (область 3). В этом окне отображается структура проекта, которая может быть представлена в файловом, структурном и ресурсном виде, как показано на рисунке 8.



а) б) в)

Рисунок 8 – Окно проектной информации, где а - файловая структура, б - библиотечная структура, в - представление проекта на уровне ресурсов

- Консольное окно для ввода и вывода информации (область 4).
- Панель закладок открытых файлов (область 5).

5. Запуск процесса симуляции

Для того, чтобы запустить процесс симуляции, необходимо в окне проектов, развернуть папку «TestBench» и выполнить макрос «A8051_exp_TB_behavior.do», как показано на рисунке 9.

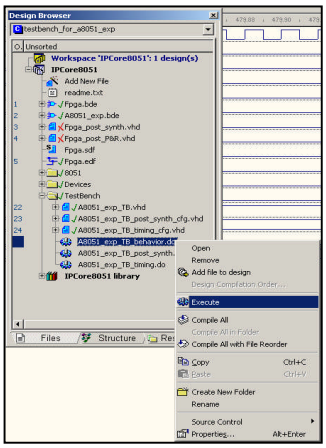


Рисунок 9 –Запуск макроса симулятора микроконтроллера

В момент проведения симуляции, среда моделирования строит временные диаграммы, которые позволяют оценить работоспособность модели ядра микроконтроллера.

На диаграмме «controller.awf», которая представлена на рисунке 10 отображается информация о состоянии 4-портов ввода вывода, значение аккумуляторного регистра, код исполняемой инструкции, значение счетчика команд.

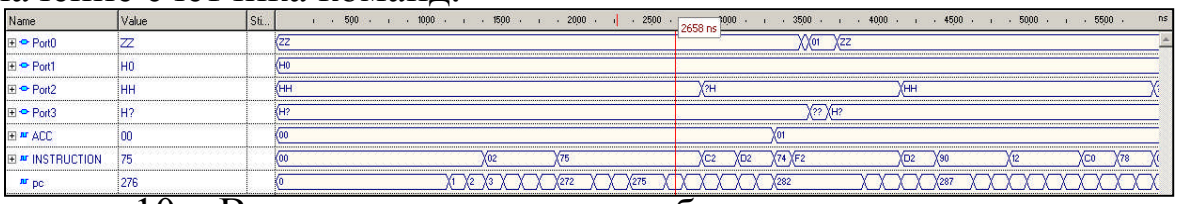


Рисунок 10 – Временная диаграмма работы микроконтроллера «controller.awf»

Временная диаграмма «display.awf» предназначена для анализа работоспособности дисплейного модуля. Пример отображения диаграммы представлен на рисунке 11.

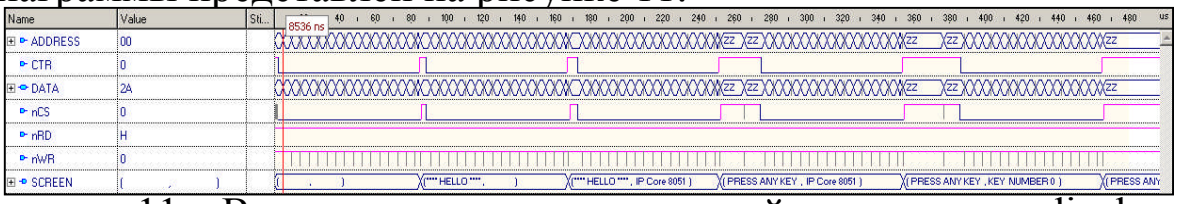


Рисунок 11 – Временная диаграмма дисплейного модуля «display.awf»

Работа клавиатурного модуля может быть проанализирована при помощи диаграммы «keyboard.awf», которая представлена на рисунке 12.

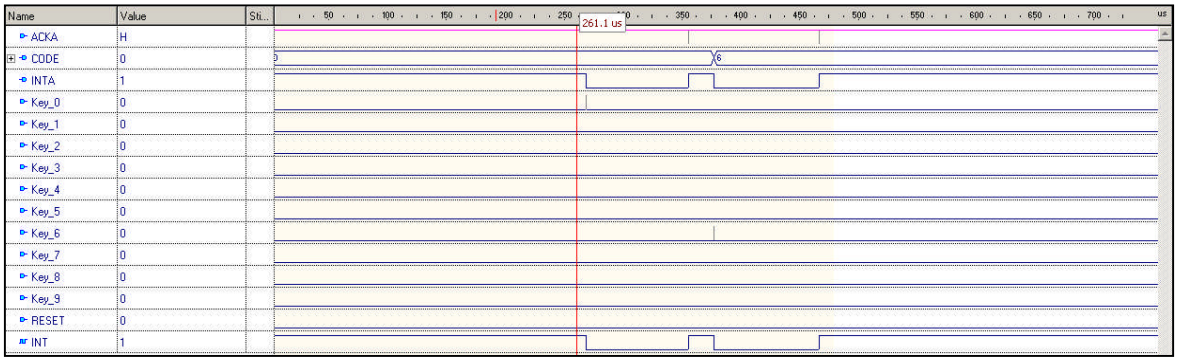


Рисунок 12 – Временная диаграмма «keyboard.awf» работы клавиатурного модуля

Работа регистра, который преобразует параллельные данные в последовательные, представлена на диаграмме «pr_sr_register.awf» и отображена на рисунке 13.

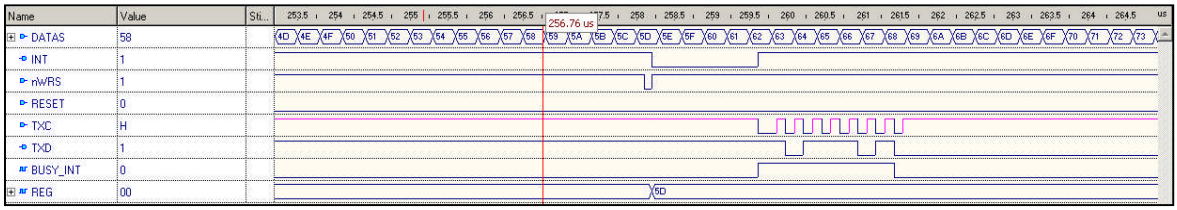


Рисунок 13 - Пример работы параллельно-последовательного регистра, представленный на диаграмме «pr_sr_register.awf»

6. Задание к лабораторной работе

- Ознакомиться с моделью микроконтроллера семейства Intel 8051.
- Изучить состав и функциональные возможности микроконтроллера.
- Исследовать среду моделирования.
- Произвести запуск симуляции модели микропроцессора.
- Получить временные диаграммы функциональных модулей модели.
- Подготовить отчет по лабораторной работе.

7. Содержание отчета.
   В отчете необходимо привести следующие сведения:

- цель лабораторной работы,
- структурная схема модели контроллера,
- краткая характеристика отдельных модулей,
- последовательность действий, необходимых для открытия проекта и запуска модели на симуляцию,
- алгоритм работы тестовой программы,
- временные диаграммы, подтверждающие работоспособность модели,
- выводы по лабораторной работе.

8. Контрольные вопросы

- Какие технические характеристики микроконтроллерного ядра семейства Intel 8051 вы можете привести?
- Из каких элементов состоит учебная модель?
- Дайте определения основным элементам модели, опишите назначение.
- Какие действия необходимо произвести, чтобы открыть проект?
- Какие действия необходимо произвести, чтобы запустить модель ядра на симуляцию?
- Какой алгоритм работы тестовой программы модели?
- Какие данные, отображаемые на временной диаграмме, подтверждают работоспособность модели?
- Какие сообщения отображаются в консольном окне, во время симуляции работы модели микроконтроллера?

-

9. Список литературы

1. Белов В. И. Проектирование цифровых диагностических систем на примере разработки генератора-анализатора : Учебно-методическое пособие по дисциплине группового проектного обучения (ГПО) / В. И. Белов, А. А. Пшенников ; Федеральное агентство по образованию, Томский государственный университет систем управления и радиоэлектроники, Кафедра телекоммуникаций и основ радиотехники. - Томск : ТУСУР, 2007. - 61 с.
2. Бибило П.Н. Основы языка VHDL., М.: СОЛОН-Пресс,2000. - 200с.
3. Перельройзен Е.З. Проектируем на VHDL – М.: СОЛОН-Пресс, 2004. – 448с.

ПРИЛОЖЕНИЕ 1

Таблица 1

Команды передачи данных

Название команды	Мнемокод	КОП	Б	Ц	Операция
1	2	3	4	5	6
Пересылка в аккумулятор из регистра (п=0+7)	MOV A, R_n	1110.1rrr	1	1	(A)<-(R_n)
Пересылка в аккумулятор прямоадресуемого байта	MOV A, ad	1110.0101	2	1	(A)<-(ad)
Пересылка в аккумулятор байта из РПД (i=0,1)	MOV A, @R_i	1110.011i	1	1	(A)<-((R_i))
Загрузка в аккумулятор константы	MOV A, #data8	0111.0100	2	1	(A)<-#data8
Пересылка в регистр из аккумулятора	MOV R_n, A	1111.1rrr	1	1	(R_n)<-(A)
Пересылка в регистр прямоадресуемого байта	MOV R_n, ad	1010.1rrr	2	2	(R_n)<-(ad)
Загрузка в регистр константы	MOV R_n, #data8	0111.1rrr	2	1	(R_n)<-#data8
Пересылка по прямому адресу аккумулятора	MOV ad, A	1111.0101	2	1	(ad)<-(A)
Пересылка по прямому адресу регистра	MOV ad, R_n	1000.1rrr	2	2	(ad)<-(R_n)
Пересылка прямоадресуемого байта по прямому адресу	MOV add, ads	1000.0101	3	2	(add)<-(ads)
Пересылка байта из РПД по прямому адресу	MOV ad, @R_i	1000.011i	2	2	(ad)<-((R_i))
Пересылка по прямому адресу константы	MOV ad, #data8	0111.0101	3	2	(ad)<-#data8
Пересылка в РПД из аккумулятора	MOV @R_i, A	1111.011i	1	1	((R_i))<-(A)
Пересылка в РПД прямоадресуемого байта	MOV @R_i, ad	0110.011i	2	2	((R_i))<-(ad)
Пересылка в РПД константы	MOV @R_i, #data8	0111.011i	2	1	((R_i))<-#data8
Загрузка указателя данных	MOV DPTR, #data16	1001.0000	3	2	(DPTR)<-#data16
Пересылка в аккумулятор байта из ПП	MOVC A, @A+DPTR	1001.0011	1	2	A<-((A)+(DPTR))
Пересылка в аккумулятор байта из ПП	MOVC A, @A+PC	1000.0011	1	2	(PC)<-(PC)+1, (A)<-((A)+(PC))
Пересылка в аккумулятор байта из памяти данных	MOVX A, @Ri	1110.001i	1	2	(A)<-((Ri))
Пересылка в аккумулятор байта из расширенной памяти данных	MOVXA,@DPTR	1110.0000	1	2	(A)<-((DPTR))
Пересылка в память данных значение из аккумулятора	MOVX @R_i, A	1111.001i	1	2	((R_i))<-(A)
Пересылка в расширенную память данных значение из аккумулятора	MOVX @DPTR,A	1111.0000	1	2	((DPTR))<-(A)
Загрузка в стек	PUSH ad	1100.0000	2	2	(SP)<-(SP) + 1, ((SP))<-(ad)

Продолжение табл. 1

1	2	3	4	5	6
Извлечение из стека	POP ad	1101.0000	2	2	(ad)<-(SP), (SP)<-(SP) – 1
Обмен аккумулятора с регистром	XCH A, R_n	1100.1rrr	1	1	(A)<->(Rn)
Обмен аккумулятора с прямоадресуемым байтом	XCH A, ad	1100.0101	2	1	(A)<->(ad)
Обмен аккумулятора с байтом из памяти данных	XCH A, @R_i	1100.011i	1	1	(A)<->((R_i))
Обмен младших тетрад аккумулятора и памяти данных	XCHD A, @R_i	1101.011i	1	1	(A_0…A_3)<->W(R_{i0}…R_{i3})

Таблица 2

Арифметические операции

Название команды	Мнемокод	КОП	Б	Ц	Операция
1	2	3	4	5	6
Сложение аккумулятора с регистром (n= 0…7)	ADD A, R_n	0010.1rrr	1	1	(A)<-(A) + (R_n)
Сложение аккумулятора с прямоадресуемым байтом	ADD A, ad	0010.0101	2	1	(A)<-(A) + (ad)
Сложение аккумулятора с байтом из памяти данных (i = 0,1)	ADD A, @R_i	0010.011i	1	1	(A)<-(A) + ((R_i))
Сложение аккумулятора с константой	ADD A, #data8	0010.0100	2	1	(A)<-(A) + #data8
Сложение аккумулятора с регистром и переносом	ADDC A, R_n	0011.1rrr	1	1	(A)<-(A) + (R_n) + (C)
Сложение аккумулятора с прямоадресуемым байтом и переносом	ADDC A, ad	0011.0101	2	1	(A)<-(A) + (ad) + (C)
Сложение аккумулятора с байтом из памяти данных и переносом	ADDC A, @R_i	0011.011i	1	1	(A)<-(A) + ((R_i)) + (C)
Сложение аккумулятора с константой и переносом	ADDC A, #data8	0011.0100	2	1	(A)<-(A) + #data8 + (C)
Десятичная коррекция аккумулятора	DAA	1101.0100	1	1	Если (A_0…A_3)>9 или ((AC)=1), то (A_0…A_3)<-(A_0…A_3)+6, затем если (A_4…A_7)>9 или ((C)=1), то (A_4…A_7) <– (A_4…A_7)+6
Вычитание из аккумулятора регистра и заёма	SUBB A, R_n	1001.1rrr	1	1	(A)<-(A) – (C) – (R_n)
Вычитание из аккумулятора прямоадресуемого байта и заема	SUBB A, ad	1001.0101	2	1	(A)<-(A) – (C) – ((ad))
Вычитание из аккумулятора байта памяти данных и заема	SUBB A, @R_i	1001.011i	1	1	(A)<-(A) – (C) – ((R_i))
Вычитание из аккумулятора константы и заема	SUBB A, data8	1001.0100	2	1	(A)<-(A) – (C) – #data8

Продолжение табл. 2

1	2	3	4	5	6
Инкремент аккумулятора	INC A	0000.0100	1	1	(A)<-(A) + 1
Инкремент регистра	INC R_n	0000.1rrr	1	1	(R_n)<-(R_n)+ 1
Инкремент прямоадресуемого байта	INC ad	0000.0101	2	1	(ad)<-(ad) + 1
Инкремент байта в памяти данных	INC @R_i	0000.011i	1	1	((R_i))<-((R_i))+1
Инкремент указателя данных	INC DPTR	1010.0011	1	2	(DPTR)<-(DPTR) + 1
Декремент аккумулятора	DEC A	0001.0100	1	1	(A)»-(A)-1
Декремент регистра	DEC R_n	0001.1rrr	1	1	(R_n)<-(R_n)-1
Декремент прямоадресуемого байта	DEC ad	0001.0101	2	1	(ad)<-(ad)-1
Декремент байта в памяти данных	DEC @R_i	0001.011i	1	1	«R_i))<-((R_i))-1
Умножение аккумулятора на регистр В	MUL AB	1010.0100	1	4	(B)(A)<-(A)*(B)
Деление аккумулятора на регистр В	DIV AB	1000.0100	1	4	(A).(B)<-(A)/(B)

Таблица 3

Логические операции

Название команды	Мнемокод	КОП	Б	Ц	Операция
1	2	3	4	5	6
Логическое И аккумулятора и регистра	ANL A, R_n	0101.1rrr	1	1	(A)<-(A) AND (R_n)
Логическое И аккумулятора и прямоадресуемого байта	ANL A, ad	0101.0101	2	1	(A)<-(A) AND (ad)
Логическое И аккумулятора и байта из памяти данных	ANL A, @R_i	0101.011i	1	1	(A)<-(A)AND((R_i))
Логическое И аккумулятора и константы	ANL A, #data8	0101.0100	2	1	(A)<-(A) AND #data8
Логическое И прямоадресуемого байта и аккумулятора	ANL ad, A	0101.0010	2	1	(ad)<-(ad) AND (A)
Логическое И прямоадресуемого байта и константы	ANL ad, #data8	0101.0011	3	2	(ad)<-(ad) AND #data8
Логическое ИЛИ аккумулятора и регистра	ORLA, R_n	0100.1rrr	1	1	(A)<-(A) OR (R_n)
Логическое ИЛИ аккумулятора и прямоадресуемого байта	ORLA, ad	0100.0101	2	1	(A)<-(A) OR (ad)
Логическое ИЛИ аккумулятора и байта из памяти данных	ORLA, @R_i	0100.011i	1	1	(A)<-(A) OR ((R_i))
Логическое ИЛИ аккумулятора и константы	ORL A, #data8	0100.0100	2	1	(A)<-(A) OR #data8
Логическое ИЛИ прямоадресуемого байта и аккумулятора	ORL ad, A	0100.0010	2	1	(ad)<-(ad) OR (A)
Логическое ИЛИ прямоадресуемого байта и константы	ORL ad, #data8	0100.0011	3	2	(ad)<-(ad) OR #data8
Исключающее ИЛИ аккумулятора и регистра	XRL A, R_n	0110.1rrr	1	1	(A)<-(A) XOR (R_n)

Продолжение табл. 3

1	2	3	4	5	6
Исключающее ИЛИ аккумулятора и прямоадресуемого байта	XRL A, ad	0110.0101	2	1	(A)<-(A) XOR (ad)
Исключающее ИЛИ аккумулятора и байта памяти данных	XRL A, @R_i	0110.0111	1	1	($_(A)$)<-(A) XOR ((R_i))
Исключающее ИЛИ аккумулятора и константы	XRL A, #data8	0110.0100	2	1	(A)<-(A) XOR #data8
Исключающее ИЛИ прямоадресуемого байта и аккумулятора	XRL ad, A	0110.0010	2	1	(ad)<-(ad) XOR (A)
Исключающее ИЛИ прямоадресуемого байта и константы	XRL ad, #data8	0110.0011	3	2	(ad)<-(ad) XOR #data8
Сброс аккумулятора	CLR A	1110.0100	1	1	(A)<-0
Инверсия аккумулятора	CPL A	1111.0100	1	1	(A)<-NOT(A)
Сдвиг аккумулятора влево циклический	RL A	0010.0011	1	1	(A_{n+1})<-(A_n), n=0…6, (A_0) <– (A_7)
Сдвиг аккумулятора влево через перенос	RLC A	0011.0011	1	1	(A_{n+i})<-(A_n), n=0…6, (A_0) <– (C), (C) <-A_7)
Сдвиг аккумулятора вправо циклический	RR A	0000.0011	1	1	(A_n)<-(A_{n+}i), n=0…6, (A_7)<– (A_0)
Сдвиг аккумулятора вправо через перенос	RRC A	0001.0011	1	1	(A_n)<-(A_{n+}i), n=0…6, (A_7) <– (C), (C) <-(A_0)
Обмен местами тетрад в аккумуляторе	SWAP A	1100.0100	1	1	(A_0...A_3)<->(A_4...A_7)

Таблица 4

Битовые операции

Название команды	Мнемокод	КОП	Б	Ц	Операция
Сброс переноса	CLRC	1100.0011	1	1	(C)<-0
Сброс бита	CLR bit	1100.0010	2	1	(bit)<-0
Установка переноса	SETBC	1101.0011	1	1	(C)<-1
Установка бита	SETB bit	1101.0010	2	1	(bit)<-1
Инверсия переноса	CPLC	1011.0011	1	1	(C)<-NOT(C)
Инверсия бита	CPL bit	1011.0010	2	1	(bit)<-NOT(bit)
Логическое И бита и переноса	ANL C, bit	1000.0010	2	2	(C)<-(C) AND (bit)
Логическое И инверсии бита и переноса	ANL C, /bit	1011.0000	2	2	(C)<-(C)AND (NOT(b))
Логическое ИЛИ бита и переноса	ORL C, bit	0111.0010	2	2	(C)<-(C) OR (bit)
Логическое ИЛИ инверсии бита и переноса	ORL C, /bit	1010.0000	2	2	(C)<-(C)OR(NOT(bit))
Пересылка бита в перенос	MOV C, bit	1010.0010	2	1	(C)<-(bit)
Пересылка переноса в бит	MOV bit, C	1001.0010	2	2	(bit)<-(C)

Таблица 5

Операции управления аппаратно-программными средствами МП

Название команды	Мнемокод	КОП	Б	Ц	Операция
1	2	3	4	5	6
Длинный переход в полном объеме памяти программ	LJMP ad16	0000.0010	3	2	(PC)<-ad16
Абсолютный переход внутри страницы памяти	AJMP ad11	$a_{10}a_9a_8$0.0001	2	2	(PC)<-(PC)+2, (PC_{0-10})<-ad11
Короткий относительный переход	SJMP rel	1000.0000	2	2	(PC)<-(PC)+2, (PC)<-(PC)+rel
Косвенный относительный переход	JMP @A + DPTR	0111.0011	1	2	(PC)<-(A)+(DPTR)
Переход, если аккумулятор равен нулю	JZ rel	0110.0000	2	2	(PC)<-(PC)+2, если (A)=0, то (PC)<-(PC)+rel
Переход, если аккумулятор не равен нулю	JNZ rel	0111.0000	2	2	(PC)<-(PC)+2, если (A)*0, то (PC)<-(PC)+rel
Переход, если перенос равен единице	JC rel	0100.0000	2	2	(PC)<-(PC)+2, если (C)=1, то (PC)<-(PC)+rel
Переход, если перенос равен нулю	JNC rel	0101.0000	2	2	(PC)<-(PC)+2, если (C)=0, то (PC)<-(PC)+rel
Переход, если бит равен единице	JB bit, rel	0010.0000	3	2	(PC)<-(PC)+3, если (bit)=1, то (PC)<-(PC)+rel
Переход, если бит равен нулю	JNB bit, rel	0011.0000	3	2	(PC)<-(PC)+3, если (bit)=0, то (PC)<-(PC)+rel
Переход, если бит установлен, с последующим сбросом бита	JBC bit, rel	0001.0000	3	2	(PC)<-(PC)+3, если(bit)=1, то (bit)<-0 и (PC)<-(PC)+rel
Декремент регистра и переход, если не нуль	DJNZ Rn, rel	1101.1rrr	2	2	(PC)<-(PC)+2, (R_n)<-(R_n)-1, если (R_n)<>0, то (PC)<-(PC)+rel
Декремент прямоадресуемого байта и переход, если не нуль	DJNZ ad, rel	1101.0101	3	2	(PC)<-(PC)+2, (ad)<-(ad)-1, если (ad)<>0, то (PC)<-(PC)+rel
Сравнение аккумулятора с прямоадресуемым байтом и переход, если не равно	CJNE A, ad, rel	1011.0101	3	2	(PC)<-(PC)+3.если (A)<>(ad), то (PC)<-(PC)+re1,если (A)<(ad), то (C)<-1, иначе (C)<-0
Сравнение аккумулятора с константой и переход, если не равно	CJNE A, #data8, rel	1011.0100	3	2	(PC)<-(PC)+3.если (A)*#d8, то (PC)<-(PC)+rel, если (A)<#d8, то (C)<-1, иначе (C)<-0
Сравнение регистра с константой и переход, если не равно	CJNE R_n, #data8, rel	1011.1rrr	3	2	(PC)<-(PC)+3.если (Rn)<>#d, то (PC)<-(PC)+rel, если (Rn) <#d, то (C)<-1, иначе (C)<-0
Сравнение байта с константой и переход, если не равно	CJNE @R_i, #data8, rel	1011.011i	3	2	(PC)<-(PC)+3.если ((Ri))*#d, то (PC)<-(PC)+re1,если ((Ri)) <#d, то (C)<-1, иначе (C)<-0

Продолжение табл. 5

1	2	3	4	5	6
Длинный вызов подпрограммы	LCALL ad16	0001.0010	3	2	(PC)<-(PC)+3, (SP)<-(SP)+1, ((SP))<-($PC_{0...7}$), (SP)<-(SP)+1, ((SP))<-(PC_{8-15}), (PC)<-ad16
Абсолютный вызов подпрограммы в пределах страницы	ACALL ad11	$a_{10}a_9a_8$1.0001	2	2	(PC)<-(PC)+2, (SP)<-(SP)+1, ((SP))<-($PC_{0...7}$), (SP)<-(SP)+1, ((SP))<-($PC_{8}...{}_{15}$), (PC0-10)<-ad11
Возврат из подпрограммы	RET	0010.0010	1	2	($PC_{8...15}$)<-((SP)), (SP)<-(SP)-1, ($PC_{0...7}$) -((SP)), (SP)<– SP)-1
Возврат из подпрограммы обработки прерывания	RETI	0011.0010	1	2	($PC_{8...15}$) <-((SP)), (SP)<-(SP)-1, ($PC_{0...7}$) <– ((SP)), (SP)<-(SP)-1
Пустая операция	NOP	0000.0000	1	1	(PC)<-(PC)+1

ПРИЛОЖЕНИЕ 2

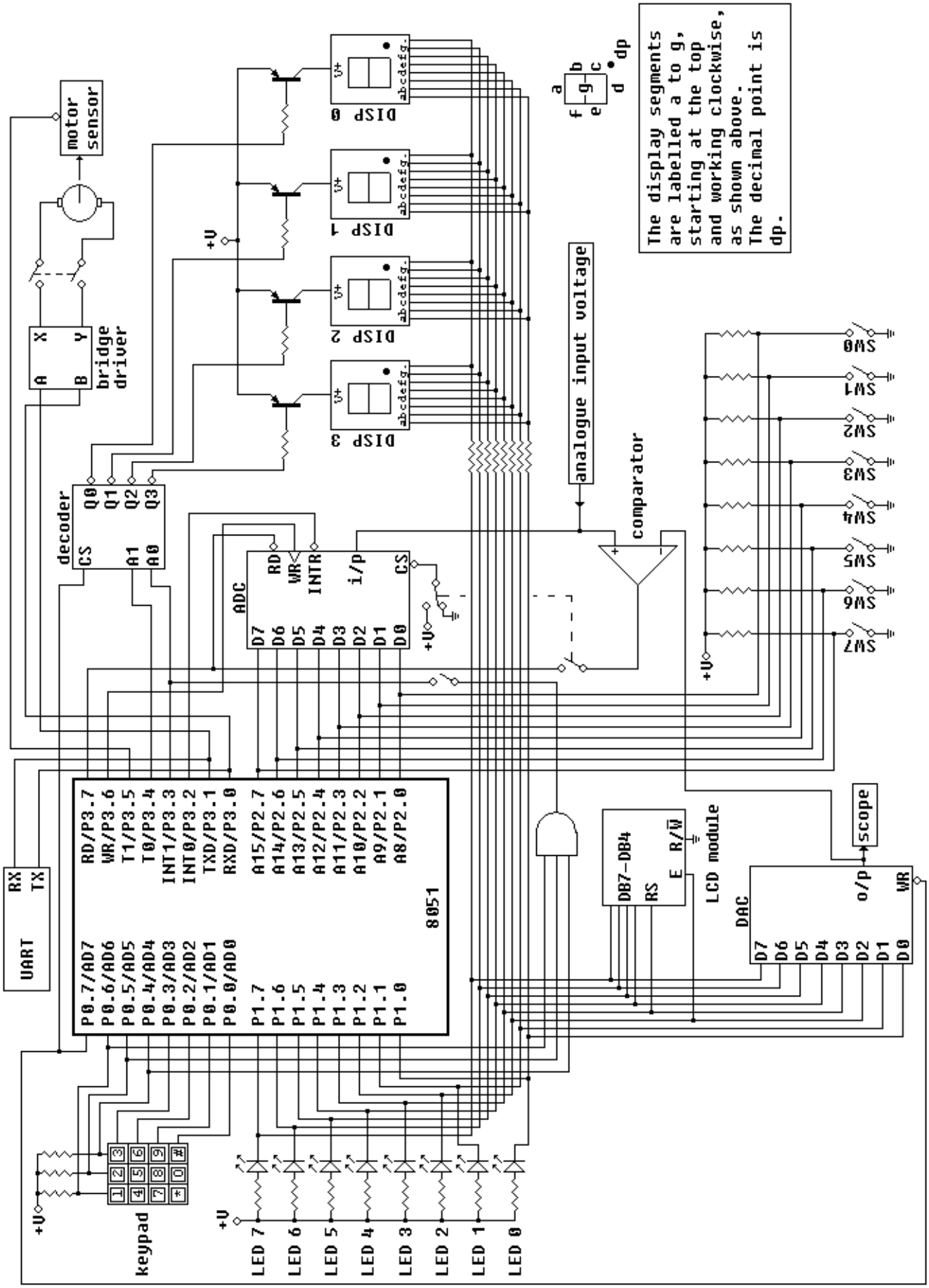


Схема электрическая принципиальная среды обучения «edsim51»