Министерство образования и науки Украины
Севастопольский национальный технический университет

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к курсовому проектированию
по дисциплине «Программирование»
для студентов направления 0915- "Компьютерная инженерия"
(дневная форма обучения)

Севастополь
2003

УДК 519.6

Методические указания к выполнению курсового проекта по дисциплине "Программирование" / Сост. А.К.Васильченко, Г.Г.Сергеев – Севастополь: Изд-во СевНТУ, 2003.–24 с. Библиогр. 5.

Целью методических указаний к курсовому проекту является закрепление у студентов навыков алгоритмизации типовых задач при практическом использовании основных разделов дисциплины «Программирование». В методических указаниях на примере разбираются этапы выполнения курсового проекта, в котором предлагается разработать и реализовать на ЭВМ программную модель учебной ЭВМ, состоящей из микропроцессора и блока памяти, разработать и отладить программу преобразования текстов программ на языке низкого уровня учебной ЭВМ в объектный код.

Методические указания рассмотрены и утвердены на заседании кафедры кибернетики и вычислительной техники (протокол № ___ от 06 июня 2003 г.) и на научно-методическом совете СевНТУ (протокол № ___ от _____).

Допущено учебно-методическим центром СевНТУ в качестве методических указаний.

Рецензенты:

Брюховецкий А.А., к.т.н, доцент кафедры КиВТ,

Бондарев В.Н., к.т.н., доцент кафедры ИС,

Тертычный А.И., ст. преподаватель кафедры КиВТ

СОДЕРЖАНИЕ

Введение

Задачи курсового проектирования по дисциплине «Программирование»:

- развитие умения разрабатывать программные модели средней сложности,
- развитие навыков выполнения научно-исследовательских работ в области разработки и стыковки многомодульных программных комплексов;
- развитие навыков работы со стандартами, научно-технической и справочной литературой в области программирования и вычислительной техники;

В методических указаниях рассматривается методика выполнения курсового проекта на современных средствах вычислительной техники; даются конкретные рекомендации по выполнению основных этапов проекта – по выбору метода решения поставленной задачи и его формализации, по разработке и реализации алгоритма. В качестве инструментальных языков для выполнения курсового проекта предлагаются языки высокого уровня Паскаль и С++. Изложены требования к оформлению курсовой проекта.

1. Основные этапы выполнения курсового проекта.

Разработка программной системы является сложным процессом, состоящим из ряда этапов жизненного цикла программного обеспечения. При выполнении курсового проекта следует изучить раздел 5.3. ДСТУ 3918-1999 [1] и соотнести его этапы с предложенными ниже.

Подготовительный этап (1-я и 2-я недели). Уточнение постановки задачи. Анализ научно-технической литературы с целью обоснования выбора метода решения. Разработка требований к программной системе, фиксирующих назначение, функциональные возможности и формы входных и выходных данных программной системы.

Проектный этап (3-я и 4-я недели). На этом этапе рассматриваются различные пути реализации поставленной задачи, предлагаются критерии оценки эффективности алгоритма и оценка с их помощью различных вариантов решения.

Реализационный этап (с 5-ой по 10-ю неделю). В начале этого этапа вырабатывается наиболее рациональное, с точки зрения студента, решение по машинной реализации модели системы и составляется график дальнейшей работы, в ходе которой необходимо запрограммировать алгоритм на выбранном языке программирования, выполнить его отладку, получить результаты и проанализировать их.

Оформительский этап (11-я и 12-я недели). На данном этапе выполняется оформление пояснительной записки в соответствии с требованиями стандартов Украины к оформлению технической документации.

Заключительный этап (13-14 недели). На этом этапе проводится защита курсовых проектов. Студент обязан представить полностью оформленную пояснительную записку к курсовому проекту за два дня до защиты. Защита проводится в форме научного доклада. Доклад должен в сжатой форме освещать постановку задачи, анализ состояния изучаемого вопроса, выбор и обоснование метода решения, обоснование структуры программы, анализ полученных результатов.

2. Разработка программных моделей вычислительных систем.

2.1. Общие принципы построения программных моделей.

Как известно, модель объекта или явления создается для того, чтобы заменить собой сам объект или явление при проведении исследования. Модель не является копией объекта (явления), она проще за счет того, что несущественные для исследования элементы, связи, функции, особенности физической реализации отброшены. Поскольку модель не равна объекту, возникает вопрос об *адекватности* модели, или о степени соответствия модели и объекта. Чтобы упростить решение этого вопроса, потребуем выполнения четырех важных принципов.

1. Принцип пространственного соответствия означает, что каждому существенному для исследования элементу объекта должен соответствовать элемент программной модели.

2. Принцип структурного соответствия означает, что элементы программной модели должны быть «соединены» так же, как соответствующие элементы объекта.

3. Принцип соответствия во времени требует, чтобы последовательность изменения состояний элементов программной модели не противоречила бы последовательности изменения состояний элементов объекта. Другими словами, если в объекте событие А предшествовало событию Б, и Б зависит от А, то и в модели событие А должно предшествовать событию Б. Если же события А и Б объекта причинно не связаны, то в программной модели события А и Б могут быть реализованы в произвольном порядке.

4. Принцип функционального соответствия требует, чтобы функция, имеющаяся в объекте, была бы представлена соответствующей функцией или цепочкой операторов в программной модели.

Программно-логическая модель микроЭВМ должна исполнять тестовую программу функционально так же, как и сама микроЭВМ – тогда мы будем считать ее адекватной.

2.2. Типовая структура микропроцессора и его функционирование.

Рассмотрим некоторые типовые электронные компоненты, которые входят в состав микропроцессора и ряда других устройств цифровой техники.

Триггер – запоминающий элемент с двумя устойчивыми состояниями. Одно состояние обозначается как "0", другое как "1". Состояния триггера могут изменяться под действием управляющего сигнала. Без управляющего сигнала состояние триггера не изменяется – он "помнит" свое состояние. Триггер хранит 1 бит информации, или одноразрядное двоичное число.

В программной модели триггер представляют булевской переменной или переменной целого типа, представляющей булевское значение[1].

---

[1] Как в языках C/C++.

Регистр – массив триггеров. Количество элементов такого массива называют разрядностью регистра. Регистр из n разрядов имеет число состояний 2^n и может хранить n-разрядное двоичное число. Данные из регистра доступны постоянно. Для изменения данных в регистре необходимо выполнить действие "запись в регистр". При таком действии данные из источника копируются в регистр. Источником данных обычно служит другой регистр. Для записи в регистр специальных констант имеются специальные действия, например, "записать все нули" или "записать все единицы". Регистры-счетчики имеют операцию "уменьшить/увеличить на 1".

В программной модели регистр представляется переменной целого типа с разрядностью, не меньшей разрядности регистра. Чтобы гарантировать, что в неиспользуемых разрядах находятся нули, значение можно поразрядно умножить (AND) на маску. Например, для байтового регистра, int reg=value & 0xFF.

Оперативная память – массив из большого количества триггеров. Оперативную память (далее – ОП) удобно представлять как двумерный массив, в котором строку называют *словом*. Количество триггеров в слове называют *разрядностью* памяти. Количество слов массива – это *объем* памяти. Слова перенумерованы, начиная с 0. Номер памяти называют *адресом*. Исходя из технических соображений, объем памяти выбирают равным целой степени числа 2. Если объем памяти равен 2^n, адрес любого слова может быть задан n-разрядным двоичным числом.

Как техническое устройство, кроме массива триггеров, ОП содержит еще два регистра – n-разрядный регистр адреса РА и регистр данных РД, разрядность которого равна разрядности памяти. Эти два регистра являются *интерфейсом* памяти, связывающим ее с внешним миром.

В этот интерфейс входят также команды *чтение* и *запись*. По команде *чтение* в РД из памяти копируется байт, адрес которого содержится в РА. По команде *запись* информация из РД копируется в слово памяти, адрес которого указан в РА.

Для чтения данных следует поместить адрес в РА памяти, и передать памяти *код* команды "чтение". В регистре РД появится копия слова с заданным адресом.

Для записи данных следует заполнить оба регистра, РА и РД, и после подачи кода команды "запись" содержимое слова с заданным адресом будет заменено копией содержимого РД.

Когда память не выполняет ни чтение, ни запись, она хранит данные. При этом регистры РА и РД сохраняют свое содержимое.

В программной модели память представляют одномерным массивом.

Шина – система проводников, соединяющая электронные элементы друг с другом. В ряде случаев к шинам относят и специальные устройства, управляющие соединением. Шины используются для передачи данных (сигналов) от одних элементов (например, регистров), к другим.

В программной модели шины обычно не представляют, но информацию о том, какие устройства соединяет шина, используют для записи операторов присваивания,

изменяющих значения регистров. Если два элемента процессора не связаны шиной непосредственно, между ними передача данных "за одно действие" невозможна.

Процессор, как технический блок, содержит ряд элементов:

Общие регистры (регистры общего назначения, РОНы) и *специальные* регистры. РОНы доступны программисту как операнды *команд* процессора. РОНы различают или по номерам (R0, R1,…) или по именам (A, B, C, …). Специальные регистры – это регистры, назначение которых в процессоре зафиксировано. Каждый специальный регистр имеет свое обозначение. Примеры специальных регистров – *регистр счетчика адреса команды* СчАК, *регистр команды* РК, *регистр флагов* РФ.

Арифметико-логическое устройство (АЛУ) – электронная схема, предназначенная для выполнения операций. АЛУ имеет два набора входных контактов – для первого и второго операндов; набор выходных контактов для результата операции, дополнительные выходные контакты для сигнализации о некоторых свойствах результата, например, "результат равен нулю" или "результат отрицательный"; управляющие входные контакты, на которые поступают сигналы, соответствующие выполняемой команде.

Устройство управления (УУ) – электронное устройство, на входные контакты которого поступают сигналы из регистра команд, а с выходных контактов подаются управляющие сигналы в АЛУ и другие элементы процессора.

Указанные элементы процессора соединяются друг с другом системой шин.

В простейшем случае процессор выполняет команду так: код команды, находящийся в РК, воспринимается устройством управления, которое посылает сигналы управления в АЛУ. АЛУ, считая, что операнды для операции расположены в регистрах Р1 и Р2, получает результат и размещает его в регистре РР. Для результата формируются биты признаков (условий) и помещаются в регистр РФ.

На этом, собственно, выполнение команды заканчивается. Однако процессор дополнительно выполняет еще *продвижение* СчАК – увеличение содержимого СчАК на величину, зафиксированную для данного кода команды.4

Код команды в данном случае, поскольку заранее известно расположение операндов и результата, должен содержать только код *операции.* В более сложном случае, когда операнды расположены в РОНах, например, в А и В, код команды, помимо указания конкретной операции, еще должен содержать номера регистров первого и второго операндов. Устройство управления в этом случае сначала пересылает операнды – из регистра А в Р1, из В в Р2, и только после этого заставляет АЛУ выполнить операцию.

Бывают и более сложные команды, в которых требуется указать, что операнды или результат должны быть расположены в ОП. Форматы более сложных команд объясняются в примере при рассмотрении учебной микроЭВМ.

Процессор спроектирован для работы в комплексе с блоком ОП. При совместной работе можно загружать из ОП в РК именно те команды и в том порядке, кото-

рые нужны для решения задачи. Исходные значения операндов и результаты выполнения команд можно тоже хранить в памяти.

Команды могут иметь разную длину. Каждая команда занимает несколько (обычно 1 – 4) соседних слов памяти. Выполнение команды начинается с чтения из памяти первого слова команды и помещения его в РК. УУ процессора по коду команды определяет, нужно ли будет читать следующие слова. При необходимости эти слова читаются из памяти, УУ обрабатывает находящуюся в них информацию, после чего АЛУ выполняет операцию. После выполнения операции и получения результата СчАК "продвигается" на число слов выполненной команды.

Команды процессора и "продвижение" СчАК согласованы в том смысле, что команды в памяти можно записывать подряд, без пропусков. После завершения текущей команды в СчАК всегда находится адрес первого слова следующей команды.

Такое соединение процессора и памяти позволяет выполнять команды автоматически одну за другой. Это и есть микроЭВМ, способная выполнять программу решения задачи, хранящуюся в памяти.

ЭВМ выполняет команды независимо друг от друга. Передача информации от одной команды к другой происходит через "почтовый ящик" – один из регистров процессора или ОП.

В памяти могут храниться не только команды, но и данные. В некоторых вариантах компьютеров для программы и для данных используются различные блоки памяти. Когда ЭВМ читает команду, она обращается к блоку памяти команд. Когда читаются или записываются данные, обращение происходит к памяти данных.

3. Пример. Учебная микроЭВМ.

Далее рассмотрим пример, в котором будет задана схема микроЭВМ, приведены количество, обозначения и разрядность регистров, указан объем и разрядность памяти, перечислен набор операций и команд процессора.

В состав процессора входят:

- арифметико-логическое устройство (АЛУ) для выполнения операций, указанных в таблице 1.
- регистры АЛУ: регистры операндов Р1 и Р2, регистр результата РР, регистр флагов РФ;
- регистры общего назначения (РОНы) A,B,C,D или R0..R3;
- регистр команды РК;
- счетчик адреса команд (СчАК);
- устройство управления УУ;
- шина данных ШД, шина адреса ША и шина управления ШУ.

Шина ШД связывает регистры Р1, Р2, РР, РК, РД. Шина ША связывает СчАК, РК, РА. Процессор имеет внутреннюю шину данных, которая связывает регистры Р1, Р2, РР и РОНы. Выполнение программы начинается с байта 01.

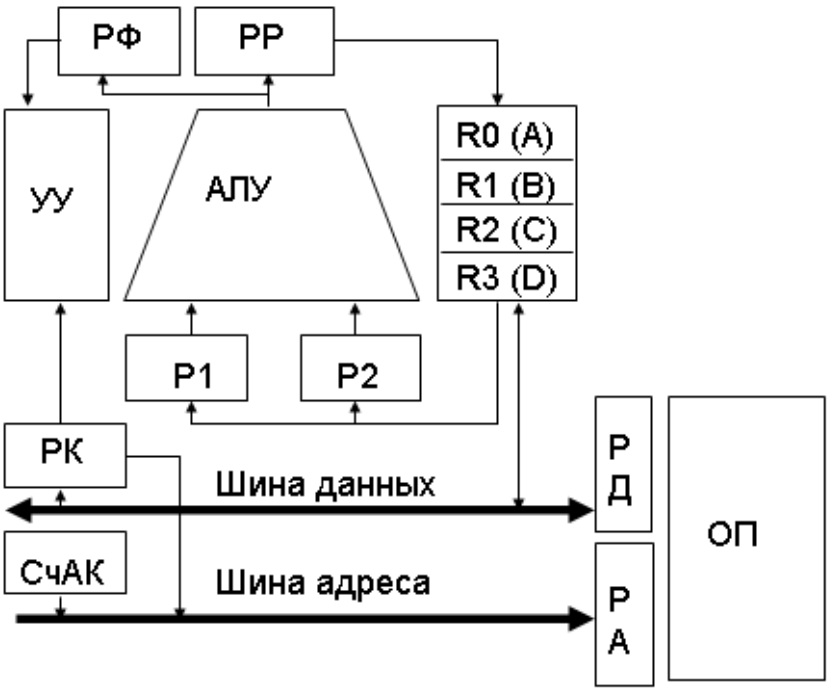

Рисунок 1. Структура учебной микроЭВМ.

Элементы РФ (регистр флагов), РР (регистр результата), регистры общего назначения (A, B, C, D), регистр команд РК, счетчик адреса команды СчАК вместе с устройством управления УУ составляют собственно микропроцесоор. Регистр адреса РА, регистр данных РД и массив запоминающих элементов ОП образуют блок оперативной памяти (внутренние связи элементов ОП не показаны). Шина данных является двунаправленной шиной, позволяющей передавать данные в любом из двух возможных направлений. При чтении команды Данные из РД оперативной памяти передаются в РК. При выполнении команды LOAD данные из РД передаются в один из РОНов. При выполнении команды STORE данные из РОН передаются в РД. Шина адреса – однонаправленная. Данные из СчАК и РК могут передаваться в РА памяти. Первая передача происходит при необходимости прочесть команду, вторая – при чтении операнда из памяти.

Все регистры процессора содержат по восемь бит. Разрядность памяти – 8 бит, объем памяти – 256 байтов.

Система команд учебной микроЭВМ приведена в таблице 1.

Таблица 1 – Мнемоника команд учебного микрокомпьютера

биты $d_0d_1d_2d_3$	Длина	Мнемокод	Мод.РФ[1]	Пояснение	fQ_1Q_2
0000 (0)	1	ADD рег1, рег2	да	рег1=рег1+рег2	001
0001 (1)	1	AND рег1, рег2	да	рег1=рег1&рег2	001
0010 (2)	1	OR рег1, рег2	да	рег1=рег1|рег2	001
0011 (3)	1	NOT рег	да	инверсия битов регистра	010
0100 (4)	2	LOAD рег, адрес	нет	загрузка данных в регистр из памяти	101
0101 (5)	2	STORE рег, адрес	нет	запись данных из регистра в память	101
1000 (8)	2	JMP адрес	нет	безусловный переход по адресу	110
1001 (9)	2	JZ адрес	нет	переход по адресу, если 0	110
1010 (A)	2	JNZ адрес	нет	переход по адресу, если не 0	110
1011 (B)	2	JL адрес	нет	переход по адресу, если меньше	110
1100 (C)	1	CLC	да	сброс битов РФ в 0	000
1101 (D)	1	HALT	нет	останов	000
1111 (F)	1	NOP	нет	нет операции	000

Число команд N_k = 13. Для различения этих команд достаточно иметь $L_k=\lceil \log_2 N_k \rceil$ = 4 бита. Четыре бита дают 16 комбинаций, 13 из них выбраны для обозначения команд, а три не используются. Если в РК попадет один из незадействованных кодов, процессор его пропускает, т.е. *выполняет как команду NOP*.

Команды могут занимать в памяти один или два байта. Однобайтные команды или не имеют операндов (как CLC, HALT, NOP), или имеют один (как NOT) или два (как ADD, AND, OR) регистровых операнда. В двухбайтных командах второй (или единственный) операнд является адресом слова памяти.

Почему нет команд с тремя операндами, как в типичном действии $a=b+c$? В командах с двумя регистровыми операндами из 8 битов 4 бита занимает код опера-

[1] Словом "да" отмечены команды, изменяющие регистр флагов РФ.

ции, оставшиеся 4 бита делятся поровну между первым и вторым операндом. В результате в качестве и первого, и второго операндов можно указать любой из четырех РОНов (типовое кодирование такое: 00-A, 01-B, 10-C, 11-D). Для третьего регистрового операнда потребовалось бы еще два бита, т.е. команда стала бы двухбайтной. Но "к.п.д." использования битов был бы низкий: 10/16*100%, и скорость выполнения таких команд понизилась бы из-за того, что теперь пришлось бы дважды обратиться к памяти, чтобы прочитать два слова команды.

Предположим, что мы придумали команду с тремя операндами, каждый из которых задан адресом. Команда состояла бы из четырех байтов с кпд использования битов = (4+8+8+8)/32*100%. Выполнение команды потребовало бы семикратного обращения к памяти – 4 обращения, чтобы прочесть все слова команды, и три обращения, чтобы прочесть (или записать) значения операндов. Выполнения такого же действия с имеющимися командами потребовало бы 7 байтов для команд (LOAD, LOAD, ADD, STORE) и девяти обращений к памяти. Казалось бы, выигрыш очевиден. Однако при вычислении $a=b+c+d$ выигрыша уже бы не было. Поэтому трехадресные команды используются редко.

Можно было бы придумать и другие команды. Рассмотрим такую цепочку: данные → адрес → адрес адреса → адрес адреса адреса. Пусть данные – это число 139, пусть оно хранится в памяти по адресу 121, и пусть по адресу 73 хранится число 121. В команде можно было бы указать и 139 (*непосредственный* операнд), и 121 (*прямой* адрес), и 73 (*косвенный* адрес). Конечно, дополнительно в команде придется указывать, какая ситуация имеет место.

В нашем компьютере разрядность данных – 8 битов (равна разрядности регистров процессора), разрядность адреса памяти – 8 битов, разрядность "адреса" РОН – 2 бита. Предположим, что мы хотим в команде задать только один операнд. Если это данные, т.е. *константа*, требуется 8 битов. Если это адрес памяти, то тоже требуется 8 битов, если же адрес регистра – то 2 бита. Можно обозначить ситуации разными комбинациями битов, например, 00 – операнд в РОНе, 01 – константа, 10 – прямой адрес, 11 – косвенный адрес, а ситуацию "адрес адреса адреса" исключить. Эти дополнительные служебные биты должны были бы присутствовать для каждого операнда; вместе с обозначением операции они создавали бы код команды.

Чем более разнообразен набор команд, тем проще программисту составлять программу. Но, как показано выше, некоторые команды могут понизить эксплуатационные характеристики процессора. При проектировании процессора приходится искать компромисс между быстродействием, сложностью и разнообразием команд. В результате список команд получается не очень большим – в нашем примере всего 13 команд.

Работа процессора состоит из последовательности элементарных действий:

- Чтение команды из памяти и помещение ее в регистр команд (читается первый байт команды; по ее коду видно, требуется ли читать второй байт команды);
- Определение по коду команды места нахождения данных (РОНы, рабочие регистры или память);
- Загрузка данных из памяти или из РОНов в регистры АЛУ Р1 и Р2;

- Выполнение операции в АЛУ и формирование РР и РФ (если операция не формирует РФ, его биты сохраняют старое значение);
- Запись результата в память или РОН;
- Увеличение СЧАК на длину команды.

Автоматическая работа процессора начинается сразу после включения – при этом в СчАК заносится начальный адрес. Автоматическое выполнение команд завершается после выполнения одной из специальных команд (Стоп или Переход в состояние ожидания событий).

Настоящий компьютер имеет более сложное устройство, чем приведенное на рис.1, в частности, содержит более мощный микропроцессор, средства для работы с *внешними устройствами* – жестким и гибким дисками, клавиатурой, дисплеем и т.д.

3.1. Система команд процессора и двоичный код

Двоичная программа (двоичный код) – это команды процессора в том виде, в котором они хранятся в оперативной памяти. Байты программы располагаются друг за другом в порядке возрастания адресов. Выполняются команды в *естественном* порядке, т.е. друг за другом, за исключением нескольких специальных случаев. Именно естественный порядок выполнения команд связан с последней фазой выполнения любой команды – увеличением СчАК на длину исполненной команды. Исключение делается только для команд переходов – вместо увеличения СчАК его значение заменяется операндом команды перехода.

Команда содержит код операции и информацию о расположении операндов. Каждый код операции представлен уникальной комбинацией битов (см. Таблицу 1). В нашем процессоре используются такие способы указания операндов:

- прямая регистровая адресация (команды ADD, AND, OR, NOT) операндов[1];
- прямая регистровая адресация первого операнда, и прямая адресация второго (команда загрузки LOAD, команда записи STORE);
- Непосредственная адресация (команды переходов JMP, JZ, JNZ, JL).

Как процессор определяет, какой формат имеет команда, сколько у нее операндов и какова ее длина?

Длина L=1+ $f(d_0, d_1, d_2, d_3)$, где $f(d_0, d_1, d_2, d_3)$ – булева функция, d_i – i-й разряд кода операции. Функция "один операнд" – $Q1(d_0, d_1, d_2, d_3)$, функция "два операнда" – $Q2(d_0, d_1, d_2, d_3)$. Легко проверить, что д.н.ф. этих функций имеют вид:

$$f = d_0 d_1 + \overline{d_0}\,\overline{d_1}\ ; \quad Q1 = d_0\overline{d_1} + \overline{d_1} d_2 d_3; \quad Q2 = \overline{d_0}\ \overline{d_2} + \overline{d_0}\ \overline{d_3}\ .$$

Можно выписать д.н.ф. и для других интересующих нас функций. Сложность этих функций зависит от того, как коды операций сопоставлены операциям. Поэтому при проектировании микропроцессора требуется выбрать такие коды операций, чтобы в совокупности сложность всех функций была бы минимальна.

[1] NOT – с одним операндом, остальные команды – с двумя.

При моделировании процессора целесообразно значения нужных нам функций подготовить как массив констант. Массив unsigned char F[16] позволяет хранить значения восьми булевых функций 4-х переменных. Значения элементов массива для функций f, Q1,Q2 приведены в таблице 1 (неопределенные значения доопределены нулями). Значение f определится как F[cod]&0x4, значение Q1 – как F[cod]&0x2, значение Q2 – как F[cod]&1.

Чтобы составить двоичный код, программист обязан помнить много кодов операций (для реальных процессоров – несколько десятков и даже сотен), множество форматов команд, и должен уметь "свернуть" нужную информацию в один или два байта. Чтобы упростить программирование, для кодов команд вводят мнемонические (облегчающие запоминание) обозначения (например, Add для операции сложение, Jmp (от англ. jump) для перехода, STORE – для записи байта в ОП.

Команды классифицируют по форматам, по назначению, по сходству операций. Для каждого формата или группы команд определяют правила записи операндов в удобном и понятном для человека виде. Такой набор правил и называют языком Ассемблера[1]. Программа на языке ассемблера выглядит как последовательность строчек, в каждой строчке – одна команда процессора.

Запись каждой команды состоит из мнемонического обозначения операции (первая часть) и обозначения операндов (вторая часть). Именно эти две части используются для построения машинного кода команды. Кроме того, в каждой строке можно указывать *комментарий* – произвольный текст между ';' и концом строки.

В таком виде программа имеет довольно "человечный" вид и в то же время ее легко преобразовать в машинные команды. Во-первых, такое преобразование производится *построчно*, во-вторых, мнемокод заменяется кодом операции на основании заранее заготовленной таблицы кодов, в-третьих, часть машинной команды с операндами легко определяется по полю операндов ассемблерной команды.

Анализируя таблицу команд, можно выделить пять *форматов* команд (приведены в таблице 2). Команды одного формата имеют одинаковую структуру, т.е. одинаковое расположение и назначение битов в команде, и отличаются только исполняемой операцией. Это означает, что в программной модели анализ и выполнение команд этой группы (вплоть до выполнения самой операции) может выполнять одна и та же подпрограмма.

Рассмотрим форматы команд более подробно. Код операции кодируется в первом байте битами 0,1,2,3, а биты 4 – 7 этого байта означают:

- В арифметических и логических командах биты 4, 5 указывают РОН первого операнда, биты 6,7 – РОН второго операнда (00-A, 01-B, 10–C, 11–D);
- В командах пересылки (LOAD, STORE) биты 4,5 кодируют РОН, а биты 6,7 не используются и могут иметь произвольные значения;
- В командах передачи управления и управления процессором биты 4 .. 7 первого байта не используются и могут иметь произвольные значения.

[1] ассемблер переводится как *сборщик*

- Второй байт (если он есть) является адресом операнда или адресом перехода.

Таблица 2. Форматы команд учебного микрокомпьютера.

Формат	Байт 1 (биты 0-7)								Байт 2 (биты 0-7)								Примечание
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	разряды (биты)
Ф1	d	d	d	d	r	r	q	q									ADD,AND,OR
Ф2	d	d	d	d	r	r	X	X									NOT
Ф3	d	d	d	d	X	X	X	X									CLC,HALT
Ф4	d	d	d	d	r	r	X	X	a	a	a	a	a	a	a	a	LOAD,STORE
Ф5	d	d	d	d	X	X	X	X	a	a	a	a	a	a	a	a	JMP,JZ,JNZ и т.д.

Примечание: буква d обозначает бит кода операции, буква r – бит кода регистра первого операнда, буква q – бит кода регистра второго операнда, буквы a – биты адреса операнда. Буква X обозначает разряд, значение которого может быть произвольным – 0 или 1.

Выполнение арифметических и логических операций сопровождается установкой следующих битов в регистре флагов:

- Бит 0 равен 1, если РР содержит нулевой байт[1], и равен 0 в противном случае;
- Бит 1 равен 1, если результат последней операции – отрицательное число[2], и равен 0 в противном случае;
- Бит 2 равен 1, если произошло арифметическое переполнение[3] при выполнении команды сложения, и равен 0 в противном случае.

В качестве примера расшифруем произвольные два байта как команды процессора. Пусть байт 1 равен 0x3E, байт 2 равен 0xC6. Сначала представим эти байты цепочками битов: 0011 1110 1010 0110. По таблице команд определяем, что код 0011 соответствует команде NOP. Длина команды – 1 байт. У команды один операнд – регистр. В разрядах 1110 номер регистра – левые два бита. Они задают регистр С. Итак, первый байт представляет команду инверсии регистра С – команду NOP C. Код операции второго байта – 1010. В таблице этому коду соответствует двухбайтовая команда условного перехода JNZ адрес. Биты 0110 не влияют на выполнение команды. Поскольку длина команды – два байта, необходимо рассмотреть следующий байт, содержащий адрес перехода.

3.2. Запись команд на Ассемблере

Все команды Ассемблера делят на два класса. Первый класс – это команды, задающие значение байтов памяти. Это, во-первых, команды, соответствующие командам процессора. Мнемоника этих команд приведена в таблице. Во-вторых, это

[1] Не все команды формируют результат, например, команды передачи управления, команда загрузки, команда записи не изменяют текущие значения в РР и РФ.

[2] Данные представлены в *дополнительном* коде. "Знаковый" разряд – левый. У отрицательных чисел этот разряд равен 1.

[3] При сложении положительных чисел получен отрицательный результат или при сложении отрицательных чисел получен положительный результат

специальная команда DB (Define Byte), позволяющая задать любое значение от 0 до 255. Второй класс – это инструкции Ассемблеру о том, как строить машинную программу. Из этого класса рассмотрим только команду ORG (ORiGin – начало), операнд которой задает начальный адрес размещения машинных команд. В совокупности эти команды позволяют задать как значение, так и место каждого байта программы.

В программе на Ассемблере каждая команда записывается в отдельной строке, а отдельные части команды (код, группа операндов, комментарии) разделяются пробелом. Операнды разделяются запятой. Началом комментария считается знак "точка с запятой".

Операнды, находящиеся в памяти, указываются адресом. Адрес может задаваться в десятичной или шестнадцатеричной системах счисления. Десятичный адрес задается как десятичное число без знака в пределах допустимых значений адресов (для процессора из примера 0..255). Шестнадцатеричный адрес задается шестнадцатеричными цифрами, непосредственно после которых должна быть буква H.

Регистровый операнд обозначается буквой R и цифрой от 0 до 3 (или до 7, в зависимости от числа РОН) или буквами A,B,C,D.

Константа (операнд) в команде DB может быть задана в десятичной или шестнадцатеричной системах по правилам записи адреса. Можно задать отрицательную константу, поставив перед цифрами знак "-". В этом случае ассемблер вычислит дополнительный код константы.

3.3. Программа на Ассемблере

Обычно задача формулируется в символьных обозначениях, но, так как в нашем ассемблере символические имена областей памяти не разрешены[1], мы обязаны сопоставить переменным и константам некоторые адреса памяти – выполнить *распределение памяти*. Распределение памяти следует выполнить также для промежуточных величин и для окончательного результата. Распределение памяти следует зафиксировать в таблице со столбцами "имя переменной" и "адрес".

Для освоения системы команд процессора рекомендуется запрограммировать:

- загрузку произвольного регистра содержимым байта с заданным адресом;
- запись содержимого произвольного регистра в байт с заданным адресом;
- выполнение элементарных действий вида q:=b $ c, где $ обозначает операцию над двумя операндами, а q, b и c располагаются в РОНах или в памяти (всего восемь комбинаций);
- условный оператор (требует использования команд условного и безусловного перехода);
- оператор цикла (также требует использования команд условного и безусловного перехода).

[1] Во "взрослых" ассемблерах такая возможность обязательно имеется

После освоения этих элементарных действий можно переходить к программированию более сложной задачи.

3.4. Примеры программ на ассемблере

Задача 1. Сложить содержимое двух байтов памяти и результат поместить в память, т.е. вычислить *s:=p+q*.

Поскольку процессор складывает только числа из РОНов (см. таблицу команд), схема решения задачи такова: скопировать p и q в РОНы, выполнить сложение, записать результат в память. Кратко это представим так: R0:=p; R1:=q; R0:=R0+R1; s:=R0. Для указанных действий подберем команды: LOAD R0,p; LOAD R1,q; ADD R0,R1; STORE R0,s; HALT. Выберем адреса для p, q, s: 20H, 21H и 22H.

Программа:

```
ORG 01
LOAD R0,20H; загрузка в РОН 0 шестнадцатеричной константы 12 – значения p
LOAD R1,21H; загрузка в РОН 1 шестнадцатеричной константы 3A – значения q
ADD R0,R1; сложение значений из РОН 0 и РОН 1. Результат – в РОН 0
STORE R0,22H; запись значения из РОН 0 в байт с адресом 22H
HALT; остановка автоматического выполнения команд
ORG 20H
DB 12H; значение p
DB 3AH; значение q;
```

В результате преобразования программы Ассемблером получится машинная (двоичная) программа

X-Адрес	X-Код	Двоичный код	Команда	Комментарии
			ORG 01	;задать адрес
01	4020	0100 0000 0010 0000	LOAD R0,20H	;R0=p
03	4421	0100 0100 0010 0001	LOAD R1,21H	;R1=q
05	01	0000 0001	ADD R0,R1	;R0:=R0+R1…
06	5022	0101 0000 0010 0010	STORE R0,22H	;s:=R0
08	00	0000 0000	HALT	
			ORG 20H	;задать адрес
20	12	0001 0010	DB 12H	;p
21	3A	0011 1010	DB 3AH	;q
				;здесь будет s

Длина нашей программы – 8 байтов, и еще три байта заняты исходными данными и результатом.

Задача 2. В некотором байте – положительное число p. В следующий байт записать число –p.

Напомним, числа представляются в дополнительном коде. Старший бит байта – знаковый. Значение 0 этого бита – признак положительного числа, значение 1 – отрицательного. Все беззнаковые числа из диапазона 0..255 разделены на две части. Числа 0..127 соответствуют положительным знаковым числам, числа 128..255 – отрицательным знаковым числам. Для положительного числа p дополнительный код симметричного ему отрицательного числа получается по формуле $q=-p=\overline{p}+1$.

Схема решения: поместить p в РОН и инвертировать его биты. В другой РОН поместить 1. К первому РОНу прибавить значение из второго РОНа. Результат записать в память. Для p и q выберем байты 22H и 23H. Число 1 поместим в байт 21H.

```
ORG 1
LOAD R0,22H; загрузили p в R0
NOP R0; инверсия
LOAD R1,21H; загрузили 1 в R1
ADD R0,R1; увеличили R0 на 1
STORE R0,23H; записали результат
HALT
ORG 21H
DB 1H; константа 1
DB 27H; значение p
DB 0H; место для значения q
```

После обработки ассемблером получим:

X-Адрес	X-Код	Двоичный код	Команда	Комментарии
			ORG 01	;задать адрес
01	4022	0100 0000 0010 0010	LOAD R0,22H	;R0=p
03	30	0011 0000	NOP R0	;
04	4121	0100 0100 0010 0001	LOAD R1,21H	;R1=1
06	01	0000 0001	ADD R0,R1	
07	5023	0101 0000 0010 0011	STORE R0,23H	
09	00	0000 0000	HALT	
			ORG 21H	;задать адрес
21	01	0000 0001	DB 1H	;1
22	27	0010 0111	DB 27H	;p
23	00	0000 0000	DB 0H	;здесь будет q

Задача 3. В памяти находится число p. Получить модуль этого числа $q=|p|$.

Как известно, если $p\geq 0$, то q=p, иначе q= -p. Схема решения: проверить отрицательность p (бит 1 РФ, команда JL), для отрицательного p получить –p и записать как результат. Для неотрицательного p скопировать его в q.

Выберем адреса для переменных и констант: p – 22H, q – 23H, число 0 – 20H, число 1 – 21H. Число 0 потребуется, чтобы сложить с ним p и сформировать признак в РФ.

Программа:

```
ORG 01H
LOAD R0,22H; R0=p
LOAD R1,20H; R1=0
ADD R0,R1; R0=p, установлены разряды РФ
JL 0AH; условный переход к команде с адресом 0A, т.е. к NOT
JMP 0EH; безусловный переход к команде с адресом 0E, т.е. к STORE
NOT R0;инверсия битов R0
LOAD R1,21H; R1=1
ADD R0,R1; R0= - p
STORE R0,23H
HALT
ORG 20H
DB 0H
DB 1H
DB -3BH; значение p
DB 00;место для q
```

X-Адрес	X-Код	Двоичный код	Команда	Комментарии
			ORG 01	;задать адрес
01	4022	0100 0000 0010 0010	LOAD R0,22H	;R0=p
03	4421	0100 0100 0010 0000	LOAD R1,20H	;R1=0
05	01	0000 0001	ADD R0,R1	
06	B000	1011 0000 0000 0000	JL 0AH	; ┌──┐
08	8000	1000 0000 0000 0000	JMP 0EH	; ┌─┐ │
0A	30	0011 0000	NOT R0	;◄──┘
0B	4421	0100 0100 0010 0001	LOAD R1,21H	; │
0D	01	0000 0001	ADD R0,R1	; │
0E	5023	0101 0000 0010 0011	STORE R0,23H	; ◄┘
0B	00	0000 0000	HALT	
			ORG 20H	;задать адрес
20	00	0000 0000	DB 00H	0;
21	01	0000 0001	DB 01H	;1
22	C5	1100 0101	DB -3BH	;p= - 0011 1011
23	00	0000 0000	DB 0H	;здесь будет q

При записи программы на нашем ассемблере особое внимание нужно уделить адресам памяти, выделенной под переменные – поскольку такое выделение произошло до составления программы, вполне вероятно, что последние команды программы займут память, отведенную переменным. В этом случае область данных для переменных следует сместить в сторону бóльших адресов, чтобы избежать наложения. Адреса в командах перехода могут быть определены только после того, как записаны все команды. Поэтому их сначала оставляют пустыми, отмечая переходы к нужным командам стрелками. После того, как записаны команды, к которым ведут стрелки, адреса этих команд становятся адресами перехода. Рекомендуется программу на ассемблере записывать в таблицу со строками следующей структуры:

Адрес	Команда	Длина команды	Примечание

Ассемблер, преобразуя текст программы, формирует два файла: файл с объектным кодом (команды процессора, значения переменных и констант) и файл листинга, который содержит исходный текст и результаты трансляции. Листинг программы приведен в таблице 3:

Файл объектного кода содержит информацию о машинном представлении программы. Рекомендуется два формата объектного файла: формат "признак – байт", и формат "признак – длина – байты.

В формате "признак-байт" объектный файл содержит четное число байтов. В каждой паре байтов первый может принимать всего два значения. Значение 00H будет признаком того, что второй байт пары – адрес размещения следующих байтов программы. Значение 01H будет признаком того, что второй байт пары – командный байт. Для задачи 2 объектный файл будет содержать такие байты (одиночное подчеркивание – адресная группа, двойное подчеркивание – командная группа):

00 01 01 40 01 22 01 30 01 41 01 21 01 01 01 50 01 23 01 00 00 21 01 01 01 27 01 00

В формате "признак - длина - байт" первый байт группы будет иметь такие же смысл и значение, как и в формате "признак - байт". Если первый байт имеет значение 01H, то следующий за ним байт задает длину (в пределах от 1 до 255) следующей за ним группы командных байтов. Для задачи 2 объектный файл в этом формате будет содержать такие байты:

00 01 01 09 40 22 30 41 21 01 50 23 00 00 21 01 03 01 27 00

Программа-загрузчик должна прочитать объектный файл, извлечь из него служебные байты, под управлением служебных байтов скопировать байты команд в моделируемую оперативную память.

4. Преобразование программы в объектный код

Программы, преобразующие текст программ с мнемокодами в файлы листинга и объектного кода, называются Ассемблерами. Ассемблер, который необходимо разработать, весьма прост ввиду ограничений на способы записи исходной программы. Перечислим кратко эти ограничения, указав тем самым, что допустимо в разрабатываемом ассемблере.

- Запрещены символические имена областей памяти (переменных и констант);
- Запрещены метки (символические имена) команд;
- Запрещены выражения и имена в поле операндов (разрешены имена РОНов);
- Запрещены относительная адресация и метки в командах перехода;
- Запрещены многосекционные программы;
- Запрещены макрогенерация, использование библиотек подпрограмм.

4.1. Преобразование программы в мнемокоде в объектный код (простейший ассемблер)

Задачи ассемблера весьма просты: преобразовать каждую команду в машинный код по известным правилам в соответствии с таблицей кодов команд.

Строки текста могут обрабатываться независимо, если известен адрес, по которому должна быть записана команда – *адрес размещения*. Начальный адрес размещения известен – его задает команда ORG. Каждая команда процессора или команда DB увеличивает адрес размещения на число занятых командой байтов.

При обработке отдельной строки текста целесообразно выделить подзадачи:

- разбиение строки на фрагменты – мнемонический код операции, операнды, комментарий;
- анализ кода операции и сопоставление ему машинного кода;
- преобразование операндов в машинную форму;
- формирование командного байта и запись его в объектный файл.

4.2. Диагностика ассемблера

При составлении и наборе программы возможны ошибки. Команду с ошибкой нельзя заменить машинным кодом. В этом случае построение полного объектного кода невозможно, и результат работа ассемблера должен быть отмечен как негодный для исполнения. В диагностике должно быть отмечено, в какой строке и по какой причине произошел отказ от генерации команды. Такая диагностика используется программистом, чтобы быстро исправить ошибки.

Документация к программе – ассемблеру должна содержать таблицу кодов ошибок со строками вида (номер ошибки; возможная причина или смысл ошибки). Приведем примерный неполный список возможных ошибок для нашего ассемблера: неизвестная мнемоника операции; неизвестный код (имя) регистра; неверный формат адреса операнда; неверная величина адреса операнда; ошибка в записи или величине константы.

4.3. Алгоритм работы ассемблера

Предлагается следующий алгоритм работы ассемблера:

1. Открыть файл с текстом программы в мнемокодах для чтения;
2. Подготовить файлы листинга (текстовый) и объектного кода (двоичный) для записи;
3. Пока не конец файла (начало цикла)
4. Читать строку S из исходного файла;

5. Разделять строку S на команду, операнды и комментарий;
6. Выполнить поиск кода команды в таблице команд. Если команды с такой мнемоникой не существует, выдать в файл листинга сообщение об ошибке.
7. Проверить, соответствуют ли операнды формату команды. Если нет, то выдать в файл листинга сообщение об ошибке. Перейти к пункту 4.
8. Сформировать код команды. Записать его в файл объектного кода;
9. Сформировать строку листинга. Записать ее в файл листинга (конец цикла).
10. Закрыть все файлы.

Приведенный алгоритм является стандартным. В работе над своим вариантом необходимо обосновать методику выполнения пунктов 4 – 9.

4.4. Программная модель микропроцессора. Кросс-отладка программ.

Программно-логические модели (ПЛМ) используются для отладки встроенного программного обеспечения специализированных систем на базе микропроцессоров. Такие системы получили название кросс-отладчиков или эмуляторов.

При разработке программно-логической модели регистрам процессора ставятся в соответствие эквивалентные по размеру ячейки памяти. Модель памяти – это обычный одномерный массив. Кросс-отладчик имитирует рабочий цикл процессора для каждой команды: выбирает из памяти команду, адрес которой хранится в СчАК; анализирует код команды и определяет количество байтов в команде и количество и расположение операндов; читает из памяти операнды-исходные данные; выполняет операцию и при необходимости формирует признаки результата, фиксируя их в РФ; изменяет значение в СчАК, готовясь к выполнению следующей команды.

Достоинство программно-логической модели состоит в том, что все этапы выполнения команды, поскольку выполняются программно, контролируются. При этом можно оценить частоту выполнения тех или иных действий, отладить вспомогательные подпрограммы, чтобы потом использовать их в реальной системе.

Ограниченность программно-логических моделей проявляется в том, что они почти непригодны для отладки систем, работающих в режиме реального времени.

Для того, чтобы проследить за ходом выполнения команд, формируют специальный текстовый файл – протокол отладки. После выполнения каждой команды информация о состоянии регистров и памяти выводится на экран и/или текстовый файл (протокол отладки). В протокол заносится и сама выполненная команда.

При разработке диалоговой кросс-системы в качестве примера допустимых средств отладки можно взять средства отладки систем Паскаль, Дельфи®, Visual Studio®.

5. Задание на курсовое проектирование.

5.1. Оформление задания на курсовое проектирование.

Задание на курсовое проектирование выдается индивидуально и фиксируется в листе задания установленной формы.

Тема: Разработка программных моделей вычислительных систем.

Исходные данные для проекта: требуется разработать а) простой ассемблер для трансляции набора команд микропроцессора в составе микроЭВМ в соответствии с вариантом задания; б) кросс-эмулятор для выполнения заданного набора команд с возможностью контроля содержимого и памяти при выполнении команд; в) систему тестовых примеров, демонстрирующих работоспособность ассемблера и кросс-эмулятора в комплексе. Инструментальный язык программирования – C++ или Паскаль (по выбору). Разрешено использовать системы визуального программирования (Дельфи® фирмы Borland International® или Visual Studio® фирмы Microsoft®).

Содержание пояснительной записки (перечень вопросов, подлежащих разработке: Введение. 1. Постановка задачи. 2. Разработка Ассемблера: анализ набора команд микропроцессора и характеристик микроЭВМ; выбор формата ассемблерной команды и допустимых форматов операндов и констант; составление списка возможных ошибок в записи ассемблерной команды; формулирование подзадач, связанных с преобразованием ассемблерной команды; выбор формы представления исходных, промежуточных (рабочих) и результирующих данных; выбор и обоснование методов решения подзадач; разработка подпрограмм и функций для решения подзадач; разработка и обоснование структуры главной программы; методика отладки и тестирования ассемблера. 3. Разработка кросс-эмулятора: анализ структуры микропроцессора и выбор моделирующих переменных; анализ набора команд и выделение общих действий при их выполнении; разработка подпрограмм и функций для выделенных действий; разработка главной программы; тестирование и автономная отладка кросс-эмулятора; разработка и отладка загрузчика; комплексная отладка и тестирование ассемблера и кросс-эмулятора. Заключение. Библиографический список. Приложение А– Текст программы ассемблера. Приложение Б – Текст программы кросс-эмулятора. Приложение В – Тестовый пример – листинг и протокол отладки.

При выполнении пояснительной записки следует руководствоваться требованиями стандарта [6].

Перечень графического материала: 1. Схема микроЭВМ; 2. Схема алгоритма ассемблера (чертеж); 3. Схема алгоритма кросс-эмулятора (чертеж).

Чертежи и плакаты выполняются на листах формата А2 в соответствии с требованиями стандартов[4].

Календарный план с указанием названий этапов приведен в разделе 1.

Общие требования к первой части – ассемблеру:

Необходимо предусмотреть возможность задания имени файла ассемблируемой программы, имен файла листинга и файла объектного кода в командной строке в качестве операндов программы-ассемблера. Диагностика в файле листинга должна обеспечивать локализацию ошибок.

Общие требования ко второй части – программе кросс-отладчика (эмулятора). Исходными данными для эмулятора должен быть объектный файл, полученный в

результате ассемблирования. Этот файл после преобразования загрузчиком должен задавать начальное значение блока памяти микроЭВМ. Программа-эмулятор должна предусматривать возможность задания начальных значений РОНов и СчАК моделируемой микроЭВМ. Результат работы эмулятора должен быть представлен как текстовый файл, содержащий протокол отладки.

В файле – протоколе отладки каждая строка должна соответствовать выполненной команде. В строке должны быть приведены: значение СчАК, команда, значения РОН, значение байта памяти, если он используется в команде. По мере выполнения программы информация в файле накапливается, а при завершении программы файл закрывается.

5.2. Варианты заданий

Для выполнения курсового проекта предлагается одна из схем микроЭВМ и подмножество команд этой микроЭВМ. Выбор схемы и подмножества команд производится по указанию преподавателя. Для построения "осмысленной" тестовой программы множество команд может быть расширено. Команды, не вошедшие в подмножество варианта, следует реализовать как команду NOP (нет операции).

В отдельных случаях по согласованию с преподавателем допускается модификация схемы микроЭВМ, изменение формата команд и мнемоники. Изменения отражаются в пояснительной записке в отдельном разделе – "Исследовательская часть". Раздел располагается перед разделом "Заключение". В этом разделе приводятся обоснования сделанных изменений, направленность этих изменений и влияние этих изменений на разработанные программы. В разделе "Постановка задачи" в качестве исходных данных приводятся результаты изменений, с указанием в отдельном пункте изменений по отношению к базовому варианту. На листе 1 графического материала приводится измененная схема микроЭВМ, а в тексте пояснительной записки на рисунке – базовая схема.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. ДСТУ 3918-1999 ІТ. Процеси життєвого циклу програмного забезпечення. Введ. 01.07.2000 –К.: Держстандарт України.–2000.–49 с.
2. Бек Л. Системное программное обеспечение. – М.: Мир, 1990.–425 с.
3. Баррон Д. Ассемблеры и загрузчики. – М.: Мир, 1974.–75 с.
4. С.Д.Погорелый, Е.Ф.Слободянюк. Программное обеспечение микропроцессорных систем: Справочник.–2-е изд., перераб. и доп.– К.: Тэхника, 1989.–С.3–134.
5. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Введ. 01.01.92. -К. :Госстандарт Украины, 1991. –26 с.
6. ГОСТ 2.105-95 ЕСКД. Общие требования к текстовым документам. Введ. 01.07.97. -К.: Госстандарт Украины, 1996. –36 с.