Министерство образования и науки Украины

Севастопольский национальный технический университет

ОПЕРАЦИОННЫЕ СИСТЕМЫ И СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Методические указания

к выполнению курсовой работы
для студентов специальности 07.080401 -
"Информационные управляющие системы
и технологии" всех форм обучения
Часть II: “Системное программирование”

Севастополь
2005

УДК 004.4

Методические указания к выполнению курсовой работы по дисциплине “Операционные системы и системное программирование” для студентов специальности 07.080401 – "Информационные управляющие системы и технологии" всех форм обучения. Часть II: “Системное программирование”/ Разраб. В.Ю. Карлусов, А.Л. Овчинников. - Севастополь: Изд-во СевНТУ, 2005. – 30 с.

Цель методических указаний: обеспечение возможности выполнения курсового проекта по дисциплине "Операционные системы и системное программирование", приобретение практических навыков построения учебного интерпретатора-отладчика команд языка ассемблера микропроцессора и выполнения экспериментальных исследований в области надёжности и эффективности созданного программного продукта.

Методические указания рассмотрены и утверждены на заседании кафедры Информационных систем, протокол № 02 от 13 октября 2005 г.

Рецензент: канд. техн. наук, доц. каф. Кибернетики и вычислительной техники, Сергеев Г.Г.

Допущено учебно-методическим центром в качестве методических указаний.

СОДЕРЖАНИЕ

1. ЦЕЛИ И ЗАДАЧИ КУРСОВОГО ПРОЕКТИРОВАНИЯ

Целью выполнения данной курсовой работы является:

1. Закрепление основных положений дисциплины "Операционные системы и системное программирование" в части синтаксического анализа и трансляции.

2. Приобретение навыков в области технологии создания программных систем, в частности, создания интерпретатора-отладчика команд микропроцессора для отладки программ данного микропроцессора на ПЭВМ.

3. Проведение исследований в области надёжности разработанного программного обеспечения.

4. Оценка эффективности разработанного интерпретатора-отладчика.

2. ВАРИАНТЫ ЗАДАНИЙ И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Выполнение курсовой работы включает следующие этапы:

1. Разработка процедур трансляции теста программы на языке ассемблера, состоящей из определенных вариантом задания команд. Для этого выполняется:

   1.1. Разработка и программная реализация процедур лексического анализа программы, написанной на ассемблере.

   1.2. Разработка и программная реализация процедур синтаксического анализа.

   1.3. Разработка и программная реализация процедур генерации машинного кода микропроцессора.

2. Разработка процедур дисассемблирования машинного кода микропроцессора.

3. Разработка процедур моделирования состояния и машинного цикла микропроцессора, включая нижеследующее:

   3.1. Моделирование состояния регистров и флагов микропроцессора и памяти.

   3.2. Моделирование операций машинного цикла микропроцессора при выполнении команд, определенных вариантом задания, с различными режимами адресации и доступа к памяти.

4. Построение интерфейса, предоставляющего пользователю следующие возможности:

   4.1. Ввод и редактирование текста программы на языке ассемблера (предусмотреть возможность загрузки текста программы из текстового файла и сохранения в файл).

   4.2. Трансляция текста программы в машинный код с выдачей пользователю диагностических сообщений в ходе лексического и синтаксического анализа с указанием места и типа ошибки (используются

процедуры, разработанные в пп. 1.1.- 1.3.). Предусмотреть возможность сохранения полученных машинных кодов в файл и последующей загрузки из файла.

4.3. Загрузка полученного в п 4.2. в программную модель микропроцессора, содержащую:

- регистры микропроцессора;
- флаги микропроцессора;
- память данных;
- память программ (машинные коды команд и соответствующие им дисассемблированные мнемоники команд и аргументов);
- порты ввода-вывода (если таковые есть).

4.4. Обеспечение функций демонстрации процесса выполнения ассемблерной программы, с возможностью ввода и редактирования содержимого регистров и флагов модели микропроцессора и памяти. Предусмотреть возможность пошагового выполнения ассемблерной программы, а также пуска/останова, с возможностью расстановки контрольных точек (BreakPoint) по машинной либо ассемблерной нотации программы. В процессе выполнения ассемблерных команд рассчитывать время, затрачиваемое на их выполнение на эмулируемом микропроцессоре.

5. В ходе выполнения пункта 3 и 4, необходимо подготовить данные выполнить расчёт оценки эффективности интерпретатора-отладчика.

6. По мере отладки программных модулей следует тщательно собирать статистику отказов программного продукта, с целью расчёта надёжности интерпретатора.

В связи со значительным объёмом задания, допускается комплексное выполнение курсового проекта. В этом случае, организуются бригады, состоящие из двух студентов. Один из студентов, в этом случае, разрабатывает транслятор языка ассемблера (вопросы, связанные с выполнением пп. №№ 1, 4.1.,4.2.), а другой – интерпретатор-отладчик (пп. №№ 2, 3, 4.3 и 4.4).

При этом каждое из лиц, вовлечённых в проектирование, индивидуально рассчитывает показатели надёжности и эффективности разрабатываемого им модуля, а комплексная оценка программного продукта осуществляется разработчиками совместно.

Студенты заочной формы обучения выбирают этапы проектирования индивидуально, предварительно согласовав свою деятельность с руководителем курсового проектирования.

Пояснительная записка может оформляться как индивидуально, так и комплексно, в случае если разработка была коллективной.

Варианты заданий представлены в таблице 1:

Таблица 1 – Варианты заданий

№ вар	Тип микропроцессора	Команды*	Директива
1	КР580ИК80(Intel 8080)	MOV; XCHG; ADI; ACI; ANA; RLC; JNC; INR; CMA	EQU
2	Zilog Z80	LD Rg,+DD; LD(ADDR) ,Rg; ADC RgP,RgP; SUB Rg,Rg; CP +DD; BIT; CPL; JZ	DB
3	K1816BE48(Intel MCS-48)	MOV; ADD; XCH; JC; RRC A; CALL	DD
4	K1816BE51(Intel MCS-51)	ADDC; MOVD; JF0; XRL; RLC A; RET	EQU
5	K1810BM86(Intel 8086)	MOV; ADD; MUL; DAA; ROL; JLE; JNG	DB
6	КР580ИК80(Intel 8080)	MVI; STA; ADD; XRA; RRC; JZ; DCR; DAA; CPL	DD
7	Zilog Z80	LD RgP,+DDDD; LD Rg,(ADDR); ADD A,(IX +D); AND; SET; SCF; DJNZ	EQU
8	K1816BE48(Intel MCS-48)	ANL; JNC; OUTL; ORLD; RETR; RL A	DB
9	K1816BE51(Intel MCS-51)	CLR; JBn ; ANLD; MOV; CALL; RR A	DD
10	K1810BM86(Intel 8086)	LEA; LDS; INC; SBB; CMP; JZ; JE	EQU
11	КР580ИК80(Intel 8080)	LDA; STAX; ADC; ORA; RAL; JP; DCX; CMC, XCHG	DB
12	Zilog Z80	LD Rg;Rg; LD RgP,(ADDR); ADD Rg,Rg; DEC RgP; OR; RES; CCF; JNZ	DD
13	K1816BE48(Intel MCS-48)	CPL; JF1; RR A; MOVX; RET; ADD	EQU
14	K1816BE51(Intel MCS-51)	DA; JMP; RRC A; MOVP; RETR; SWAP A	DB
15	K1810BM86(Intel 8086)	MOV; SUB; DIV; AAA; RCR; JNZ; JNE	DD
16	КР580ИК80(Intel 8080)	LDAX; SHLD; SUB; CMP; RAR; RLC; JM; DCR; STC	EQU
17	Zilog Z80	LD RgP;RgP; LD(IX+D),Rg; INC (IX+D); SBC Rg,Rg; CP Rg; NEG; JC; EXX	DB
18	K1816BE48(Intel MCS-48)	DEC; JBn; MOVD; CALL; DA; RL A	DD
19	K1816BE51(Intel MCS-51)	INC; JMPP; RL A; IN; RET; CPL	EQU
20	K1810BM86(Intel 8086)	LES, LEA; CMP; XOR; ROR; JP; JPE	DB

* - кроме указанных команд необходимо также реализовать команды NOP и HLT;
Rg – однобайтный регистр;
RgP – регистровая пара;

3. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

3.1. Общие сведения

Одной из актуальных задач, решаемых в рамках системного программирования, является разработка компонентов программного обеспечения, наделённых функциями синтаксического и семантического анализа. К таковым программным продуктам, в частности, относятся ассемблеры, интерпретаторы и отладчики для различных микропроцессоров.

Известный прогресс в области создания специализированных микропроцессоров объективно сопряжён с трудностями построения средств интерактив-

ного программирования и отладки прикладного и специализированного программного обеспечения, разрабатываемых для указанных микропроцессоров.

Отмеченные проблемы традиционно разрешают путём построения аппаратных или программных комплексов, которые выполняют полное или частичное моделирование функций микропроцессора. В случае полнофункционального моделирования речь идёт об эмуляции, а если моделированию подвергается, в частности, система команд микропроцессора, то моделирующую программу называют интерпретатором-отладчиком.

Интерпретатор-отладчик обычно, содержит в своём составе средства для синтаксического и семантического анализа команд микропроцессора, процедуры моделирования операций, выполняемых в ходе функционирования, эргономический интерфейс для отображения хода моделирования вычислительного процесса. Интерфейсная часть интерпретатора-отладчика, помимо пассивного отображения, как правило, предоставляет возможности оперативного вмешательства в последовательность выполнения модели и коррекцию данных, используемых ею.

3.2. Принципы построения интерпретаторов-отладчиков

Выполнение процессором одной команды ЭВМ составляет ***основной машинный цикл***, в общем случае, подразумевает следующие действия, называемые ***элементарными операциями*** [5].

1. Выборка из памяти ЭВМ по адресу, содержащемуся в регистре счётчика команд (PC, Program Counter), кода текущей операции (КОП) и помещение его в соответствующий регистр арифметического и логического устройства (АЛУ) процессора.
2. Определение на основании КОП адресной части соответствующей команды и помещение её в регистр команд АЛУ.
3. Осуществление, на основании способа адресации, предписываемого КОП, преобразования (модификации) командных адресов (указанных в команде), в исполнительные адреса.
4. Выборка из памяти ЭВМ операндов (операнда) по исполнительным адресам, и помещение их во внутренние операционные регистры АЛУ.
5. Выполнение необходимого действия над операндами – эта элементарная операция называется ***основной***.
6. Размещение результата основной операции по соответствующему исполнительному адресу.
7. Выполнение продвижения счётчика команд, которое, в принципе, может выполняться совместно с действиями 1 и 2, и требует явного осуществления при реализации команд условного и безусловного переходов.
8. Анализ особых ситуаций (переполнение разрядной сетки, равенство нулю результата и т.п.), поелику возникающих при выполнении операций, и

выполнение действий, связанных с наличием таких ситуаций: установка флагов, выработка сигналов et c.

Следовательно, для интерпретации машинного языка микропроцессора $МП_А$ на $ПЭВМ_Б$, для $ПЭВМ_Б$ должны быть разработаны программы, моделирующие элементарные операции $МП_А$.

Помимо этого, необходимо установить соответствие Ω между памятью $МП_A$ и $ПЭВМ_B$. Множество адресов S_B называется Ω - соответствующим [4] множеству адресов S_A, если: а) каждому адресу $\alpha_A \in S_A$ соответствует по закону Ω некоторый адрес $\alpha_B \in S_B$; б) каждый адрес $\alpha_B \in S_B$ оказывается поставленный в соответствие некоторому адресу $\alpha_A \in S_A$.

Аналогичное соответствие должно выполнено в отношении содержимого этих адресов и элементарных операций [4].

Если ’S – множество содержимых адресов, составляющих S, то ‘S_i ΩΘ – равно ’S_j, если S_j Ω – соответствует S_i, и содержимые соответствующих адресов Θ – равны.

Пусть элементарная операция μ устанавливает соответствия между множествами: $\grave{M}_\mu$ – множество аргументов операции и $\grave{S}_\mu$ – множество эффектов операции. Программу $Si_B(\mu_A)$ будем называть программой интерпретации элементарной операции μ_A на $ПЭВМ_B$, если при $\grave{M}_{\mu_A}$, ΩΘ – равном $\grave{M}_{Si_B(\mu_A)}$, имеет место $\grave{S}_{\mu_A}$ ΩΘ – равно $\grave{M}_{Si_B(\mu_A)}$.

Таким образом, ***для реализации процесса интерпретации и отладки*** $МП_A$ на $ПЭВМ_B$, ***необходимо разработать*** для последней ***следующий комплекс программ***.

1. Программа интерпретации основных элементарных операций $МП_A$.
2. Программа интерпретации элементарных операций $МП_A$, не являющихся основными.
3. Вспомогательных программ, осуществляющих преобразования адресных и командных пространств: а) получение адреса $ПЭВМ_B$, Ω – соответствующего заданному адресу $МП_A$ (разбор программа); б) преобразование заданного кода $МП_A$ в Θ – равный код $ПЭВМ_B$.
4. Вспомогательных программ, осуществляющих визуализацию на $ПЭВМ_B$, состояния $МП_A$ и его изменения в процессе интерпретации операций $МП_A$.
5. Вспомогательных программ, позволяющих пользователю выполнять изменение состояния модели $МП_A$ посредством взаимодействия с $ПЭВМ_B$.

В общем виде, процесс моделирования представлен схемой, приведённой на рисунке А.1. Как видно, процесс интерпретации команды состоит в последовательном моделировании основного машинного цикла. Анализ содержимого РС на принадлежность к адресному пространству и его Ω – преобразование, выборка из Ω ОЗУ выполняемой команды и её структурный “семантический“

разбор. Получение Ω – исполнительных адресов по Ω – командным, проверка получающихся адресов на корректность. Выборка из Ω – ОЗУ операндов моделируемой команды, переход к нужной программе интерпретации и моделирование её. Засылка результатов выполнения операции в Ω ОЗУ, или анализ причин, вызвавших аварийное выполнение операции, если это необходимо. Осуществление имитации продвижения РС, когда оно не выполняется в ходе выполнения неосновных операций.

Указанный процесс дополняется функциями обработки внешних прерываний и прерываний взаимодействия с вычислительной средой ПЭВМ, на которой выполняется интерпретация.

3.3. Оценка эффективности программы интерпретатора-отладчика

В качестве основного показателя качества интерпретации будет естественно использовать скорость его работы[4], то есть количество команд моделируемой ЭВМ, в данном случае, $МП_А$, выполняемое моделирующей $ПЭВМ_В$ в единицу времени. Указанная величина зависит от среднего числа операций $ПЭВМ_В$, затрачиваемых на интерпретацию одной команды $МП_А$, и от отношения средних быстродействий моделируемой и моделирующей ЭВМ.

Указанный показатель эффективности интерпретатора есть

$$Э_И = \frac{\nu \cdot n}{\sum_{i=1}^{n} L_i}, \qquad (3.1)$$

где $Э_И$ – эффективность программы интерпретатора; ν – отношение средних быстродействий моделирующей и моделируемой ЭВМ; n – количество различных команд в системе команд интерпретируемой ЭВМ; L_i – количество команд, выполняемое моделирующей $ПЭВМ_В$ при интерпретации i-ой команды.

Если известна статистика в виде относительной частоты использования отдельных команд $МП_А$ при решении отдельного класса прикладных задач G, то выражение (3.1) претерпит уточнение:

$$Э_И^G = \frac{\nu^G}{\sum_{i=1}^{n} \eta_i^G \cdot L_i}, \qquad (3.2)$$

где $Э_И^G$ – эффективность программы интерпретатора при решении указанного класса задач G; η_i^G – относительная частота исполнения команды i в программах класса G; ν^G – относительное быстродействие $МП_А$ и $ПЭВМ_В$ на классе задач G, которое определяется следующим образом:

$$\nu^G = \frac{\sum_{i=1}^{n} \frac{\eta_i^G \cdot \Sigma\tau_i}{L_i}}{\sum_{i=1}^{n} \eta_i^G \cdot t_i}, \qquad (3.3)$$

где t_i – время выполнения i-ой команды моделируемого $МП_A$; $\Sigma\tau_i$ – суммарное время программ интерпретации i-ой команды моделируемого $МП_A$ на $ПЭВМ_B$. Очевидно, что $L_i = L_i^{OC} + L_i^{HOC}$, где L_i^{OC} – количество команд, выполняемых непосредственно при моделировании основной операции, а L_i^{HOC} – для неосновных сопутствующих операций. Поэтому возможно дальнейшее усовершенствование расчётных выражений (3.1) – (3.3), которое в немалой степени определяется степенью пересечения систем команд интерпретируемой и моделирующей ЭВМ и выбором языковых средств.

3.4. Расчёт надёжности программы интерпретатора-отладчика

Обработка, анализ и оценка результатов испытаний выполняются на основании сбора статистических данных и дальнейшем использовании моделей надёжности программного продукта [6]. Для оценки надёжности функционирования программного продукта применена модель Коркорэна [10], которая учитывает динамику отладки программного изделия и просто интерпретируется:

$$P = \frac{1}{N}\left(N_0 + \sum_{i=1}^{T} k_i (n_i - 1)\right), \qquad (3.4)$$

где N_0 – число успешных испытаний в серии из N прогонов программы; T – число типов ошибок, по которым ведётся учёт; n_i – число ошибок i -го типа, выявленных за N прогонов; k_i – масштабирующий коэффициент, определяемый ступенчатой функцией

$$k_i = \begin{cases} q_i, & n_i > 0, \\ 0, & n_i = 0 \end{cases}, \qquad (3.5)$$

где q_i – вероятность устранения ошибки i - го типа.

В [6] отмечены следующая классификация типовых ошибок выполнения.

1. Ошибки вычислений.
2. Логические ошибки.
3. Ошибки ввода-вывода.
4. Ошибки в межпрограммных интерфейсах.
5. Ошибки в интерфейсах программа/вычислительная среда.

6. Ошибки в пользовательских интерфейсах.
7. Ошибки сопряжения с базой данных.
8. Ошибки, лежащие в основе изменений по требованию пользователя.
9. Ошибки инициализации базы данных.
10. Ошибки оператора.
11. Неопознанные ошибки.

Если величина показателя надёжности не удовлетворяет заданным показателям, испытания программного продукта могут быть продолжены до достижения необходимого значения.

4. ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

Основные этапы выполнения курсового проекта должны найти своё отражение в пояснительной записке, состоящей из нижеприводимых разделов. Если пояснительная записка оформляется каждым из авторов индивидуально, то ***содержательная часть*** должна быть ***адекватна выполненной работе***.

4.1. Примерная структура пояснительной записки

Аннотация.
Содержание.
Список исполнителей с указанием разделов, если имела место коллективная разработка.
Введение.
1. Постановка задачи.
2. Разработка процедур лексического анализа
    2.1. Построение минимального детерминированного конечного автомата, осуществляющего выделение символов языка ассемблера из потока литер.
    2.2. Формулировка диагностических и констатирующих сообщений, инициируемых модулем в процессе функционирования.
3. Разработка процедур семантического анализа и генерации кода.
    3.1. Построение интерпретирующего автомата для проверки синтаксиса и генерации машинного кода.
    3.2. Формулировка диагностических и констатирующих сообщений, инициируемых автоматом по мере работы.
4. Разработка процедур моделирования машинного цикла микропроцессора.
    4.1. Организация моделирования режимов адресации и доступа к памяти микропроцессора.

4.1.1. Анализ особенностей организации памяти и видов адресации моделируемого микропроцессора.
4.1.2. Разработка алгоритмов вычисления адресов.
4.1.3. Анализ потенциальных ошибок адресации, формулировка диагностических сообщений и разработка соответствующих программных модулей, нейтрализующих ошибку.
4.1.4. Описание разработанных программных модулей.
4.2. Организация моделирования операций машинного цикла микропроцессора.
4.2.1. Анализ структуры особенностей машинного цикла интерпретируемого процессора.
4.2.2. Разработка алгоритмов и программ моделирования неосновных операций
4.2.3. Анализ возникающих ошибок выполнения неосновных операций, разработка программных модулей диагностики и обработки.
4.2.4. Описание разработанных программных модулей.
4.3. Определение исходных данных для расчёта эффективности интерпретатора-отладчика.
5. Организация интерфейса пользователя
5.1. Определение функций, которые должен реализовывать интерфейс.
5.2. Структура экранных форм и иерархия меню.
5.3. Описание алгоритмов и методов, реализующих интерфейс.
6. Руководство пользователя.
7. Выполнение информационных расчётов.
7.1. Тестовые примеры правильных и не правильных входных данных.
7.2. Расчёт оценки эффективности разработанного интерпретатора.
7.3. Расчёт надёжности разработанного программного обеспечения.
Заключение.
Библиографический список.
Приложения.

4.2. Общие требования к оформлению пояснительной записки

1. Пояснительная записка к курсовой работе выполняется в машинописном исполнении.

2. При оформлении пояснительной записки необходимо соблюдать нормативные требования, предъявляемые к технической документации, изложенные в [2, 3] и других изданиях, заменяющих их.

3. Информация представляется в виде связного технического текста, снабжённого при необходимости иллюстративными материалами в виде таблиц, графиков, диаграммами или схем. Тексты программ, громоздкий иллюст-

ративный материал, вспомогательные выкладки и результаты приводятся в приложении.

4. Разделы должны содержать раскрытие того или иного из этапов проектирования.

Во введении традиционно помещается информация, связанная с актуальностью темы, разрабатываемой в курсовой работе, характеристики параграфов пояснительной записки, с указанием освещаемых в них вопросов.

В разделе “Постановка задачи” приводятся характеристики процессора, подлежащего моделированию, подмножество команд, заданных в варианте, общие требования к интерфейсу интерпретатора и т.п.

В разделе “Разработка процедуры лексического анализа” располагаются выкладки, сопровождающие проектирование и программную реализацию конечного автомата для сканирования ассемблерной программы и подготовке к генерации машинного кода, таблицы переходов, структуры используемых данных, алгоритмы поиска, процедуры обработки состояний конечного автомата.

В разделе “Разработка процедур семантического анализа и генерации кода” должны быть отражены данные об интерпретирующем автомате, выполняющем синтаксический анализ операторов, и генерацию машинного кода, включая сообщения об ошибках в анализируемых конструкциях с указанием причин, их вызвавших.

В разделе “Разработка процедур моделирования машинного цикла микропроцессора” необходимо подробно рассмотреть методы организации памяти и режимы её адресации, эффекты, возникающие в ходе выполнения отдельных операций, предусмотреть обработку ошибочных вычислительных ситуаций. При этом приводится разработка и описание алгоритмов моделирования и программ, их реализующих, согласно [3, ГОСТ 19.402-78. Описание программы, ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем].

В разделе “Организация интерфейса пользователя” рекомендуется определить и обосновать функциональные возможности, предоставляемые пользователю. Результатом этой аналитической работы должна явиться структура экранных форм и иерархия меню.

Раздел “Выполнение информационных расчётов” содержит расчёт оценки эффективности разработанного интерпретатора и расчёт надёжности разработанного программного обеспечения. В этом же разделе проводится разработка и описание тестовых примеров, руководствуясь [3, ГОСТ 19.301 – 79. Программа и методика испытаний], публикуется статистика возникающих ошибок [6], оформляемая таблично или в виде гистограмм, фиксируется общее число прогонов задачи и рассчитывается показатель надёжности Коркорэна. Рекомендуется проектировать тесты, содержащие корректные команды микропроцессора, различной прикладной направленности, что позволит получить детальную оценку показателя эффективности.

В приложение к пояснительной записке включаются тексты программ [3, ГОСТ 19.401-78. Текст программы] и описание программ [3, ГОСТ 19.402-78.

Описание программы], если последнее очень объёмно и делает основную часть пояснительной записки неудобочитаемой.

Программные документы, приведённые в пояснительной записке и приложении, должны быть оформлены в соответствии с требованиями [3, ГОСТ 19.001-77. Общие положения; ГОСТ 19.103-77. Обозначение программ и программных документов; ГОСТ 19.104-78. Основные надписи; ГОСТ 19.105-78. Общие требования к программной документации; ГОСТ 19.201-78. Требования к содержанию и оформлению; ГОСТ 19.202-78. Спецификация; ГОСТ 19.404-79. Пояснительная записка; ГОСТ 19.502-78. Описание применения; ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем].

Предупреждение: Работа, оформленная с грубыми нарушениями требований, изложенных в настоящем параграфе, и ДСТУ (ГОСТах) к защите не принимается. Незначительные отступления могут повлечь за собой снижение общей оценки, выставляемой курсовой работе.

5. ПОРЯДОК ЗАЩИТЫ КУРСОВОЙ РАБОТЫ

Курсовая работа защищается учащимся в присутствии комиссии, состоящей из преподавателей кафедры. График защиты работ доводится до студентов не позднее, чем за месяц до её начала. К защите необходимо подготовить доклад и демонстрационные материалы, отражающие основные технические решения и результаты работы. *Пояснительная записка должна быть представлена руководителю для рецензирования* ***не позднее****, чем за* ***два дня*** *до защиты.*

Во время защиты студент представляет проделанную работу в форме доклада доклад. В доклад рекомендуется построить в следующей последовательности:

– постановка задачи;
– обоснование принятых решений;
– основные этапы разработки;
– результаты расчётов оценок эффективности интерпретатора и показателей надежности его функционирования.

После окончания доклада студенты должны быть готовы к ответам на вопросы, как по существу работы, так и по основным положениям дисциплин, используемых в работе.

Студент несёт полную ответственность за все организационные, алгоритмические и программные решения, принятые в работе или используемые в ней.

6. ИЛЛЮСТРАЦИИ ОТДЕЛЬНЫХ ЭТАПОВ КУРСОВОГО ПРОЕКТА

Ниже рассматривается структура хода проектирования интерпретатора учебного микропроцессора, система команд которого представлена грамматикой в виде НФБ. Из приведённой информации следует, что микропроцессор является одноадресным, команда состоит метки, кода операции (или директивы),

из которых обязателен код операции. Имеются три способа адресации: непосредственная (#), прямая (@) и абсолютная (@#). Имеются директивы компилятору start и end, обрамляющие исполняемые участки кодов; указание адреса микропроцессора, начиная с которого следует разместить следующий фрагмент данных – org; кратковременное прерывание выполнения программы с передачей управления микропрограмме ждущего режима – pause. Причем, end приводит к генерации соответствующей команды, pause генерирует прерывание, которое можно завершить внешне, start и org в машинные команды никоим образом не распространяется.

<**Stmt**> → <**Lbl**><**Cop**><**Oprnd**>|<**Dir**><**Oprnd**>|<**Dir**>|<**Cop**>|<**Lbl**><**Dir**>
<**Stmt**> → <**Lbl**><**Dir**><**Oprnd**>|<**Lbl**><**Cop**>
<**Lbl**> → <*iden*>
<**Cop**> → *add*|*neg*|*jmp*|*shift*|*load*|*store*|*mult*|*div*
<**Dir**> → *start*|*end*|*org*|*pause*
<**Oprnd**> → #<*data*>|@<*iden*>|@#<*data*>
<*iden*> → <**Bukva**><**Next**>|<**Bukva**>
<**Bukva**> → *a*|*b*|*c*|...|*z*
<**Next**> → <**Bukva**>|<**Cifra**>|<**Bukva**><**Next**>|<**Cifra**><**Next**>
<**Cifra**> → *0*|*1*|*2*|...|*9*
<*data*> → *0*<**Bin**>*b*|*0*<**Hex**>*h*
<**Bin**> → <**Bin**><**Dig2**>|<**Dig2**>
<**Dig2**> → *0*|*1*
<**Hex**> → <**Hex**><**Dig16**>|<**Dig16**>
<**Dig16**> → *0*|*1*|*2*|...|*9*|*a*|*b*|...|*f*

Разработку процедур лексического анализа программы на ассемблере микропроцессора и подготовка к генерации машинного кода целесообразно начать с построения конечного автомата, выделяющего цепочки, соответствующие символам языка.

Эскизные регулярные выражения для описания частичных конечных автоматов суть следующие.

1. Идентификатор или метка б{б∨ц}L_1.
2.Числовая константа 0{0∨1}b∨0{0∨1∨2∨3∨4∨5∨6∨7∨8∨9∨a∨b∨c∨d∨e∨f}h.
3. Мнемонические коды (add∨neg∨jmp∨shift∨load∨store∨mult∨div) L_1.
4. Директивы компилятору (start∨end∨org∨pause) L_1.

В указанных регулярных выражениях L_1 есть литера, не являющаяся буквой или цифрой.

Их объединение, дополненное лексемой L_2, описывающей класс литер, не принадлежащих алфавиту рассматриваемого ассемблера, даёт полное регулярное выражение для описания конечного автомата, решающего задачи сканирования. Данный автомат может быть реализован известными Вам методами.

Синтаксический и семантический анализ для языков, структура предложений которых незамысловата, целесообразно производить с использованием

магазинного автомата, граф переходов которого, представленный на рисунке 6.1, соответствует синтаксическому графу грамматики [1].

Представленная схема схематически отображает ход анализа, ибо в неё не вписываются ситуации с директивами или кодами операций, не имеющими операнда. Тем не менее, появление таких схем логично в ходе проектирования, являющего собой процесс последовательного уточнения наших представлений об объекте проектирования. Каждое состояние интерпретирующего автомата, помимо опознания текущего символа, сопровождается формированием фрагмента машинного кода команды. Формальные аспекты генерации кодов достаточно подробно отображены в [1, 8, 9].

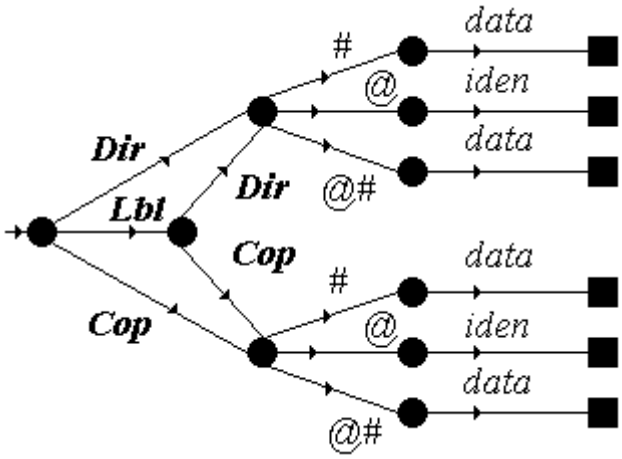

Рисунок 6.1 – Граф переходов конечного автомата при синтаксическом анализе

Моделирование машинного цикла определяется типом микропроцессора. В отдельных простых случаях, в частности, рассматриваемом в Б.1, он состоит из двух укрупнённых операций: извлечение из памяти и выполнение кода. В приведённом программном фрагменте вначале даны модели структур данных и памяти. Программный код H6809 просматривается побайтно. Предполагается, что код операции занимает байт, содержимое которого анализируется, процедурой, моделирующей выполнение, текст Pascal – программы которой приведён в Б.2. В описательной части данной процедуры приводятся вспомогательные процедуры, моделирования неосновных операций машинного цикла: считывание очередных байтов команды, установка флагов, выполнение переходов.

Если интерпретируемый процессор обладает способностью поддерживать несколько режимов адресации (как и в случае нашего демонстрационного примера), необходимо тщательно промоделировать эти режимы. Подобная модель режимов адресации иллюстрируется для микро – ЭВМ PDP – 11 в приложении Б.3. Заметим, что в функции RMW фиксируется ошибка, могущая возникнуть при чтении из памяти программного кода, минимальная длина которого для

PDP – 11 составляет 15-ти разрядное слово, и, следовательно, байтовый адрес чётен.

Одними из нетривиально моделируемых операций, выполняемыми микропроцессорами, являются операции беззнакового и знакового умножения и деления. Тексты Pascal – программ, выполняющих такие операции, помещены в приложении Б.4 – Б.7. Если необходимо выполнять операции над числами с плавающей точкой, следует различать действия над порядками и мантиссами операндов, производить выравнивание результата по разрядной сетке, осуществлять нормализацию.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Грис Д. Конструирование компиляторов для ЦВМ / Д. Грис. – М.: Мир, 1975. - 544 с.

2. ДСТУ 3008 - 95. Документация. Отчеты в сфере науки и техники. Введён 23.02.95. - К.: Госстандарт Украины, 1995. - 37 с.

3. Единая система программной документации: [Сборник]: ГОСТ 19.001-77 - ГОСТ 19.102-77 - ГОСТ 19.103-77 - ГОСТ 19.104-78 - ГОСТ 19.105-78 - ГОСТ 19.201-78 - ГОСТ 19.202-78 - ГОСТ 19.301-79 - ГОСТ 19.401-78 - ГОСТ 19.402-78 - ГОСТ 19.404-79 - ГОСТ 19.502-78 - ГОСТ 19.503-79 - ГОСТ 19.504-79 - ГОСТ 19.505-79 - ГОСТ 19.506-79 - ГОСТ 19.508-79 - ГОСТ 19.701-90. - М.: Издательство стандартов, 1990. - 447 с.

4. Косарев В.В. Об использовании метода интерпретации машинных языков при разработке автоматизированных систем отладки программного обеспечения специализированных ЦВМ / В.В. Косарев // Кибернетика. – 1977. – № 1. – С. 75 – 88.

5. Папернов А.А. Логические основы цифровой вычислительной техники / А.А. Папернов. - М.: Советское радио, 1972. - 529 с.

6. Тейер Т. Надёжность программного обеспечения. / Т. Тейер, М. Липов, Э. Нельсон. – М.: Мир, 1981. – 323 с.

7. Уокерли Дж. Архитектура и программирования микро-ЭВМ / Дж. Уокерли: в 2-х книгах. – М.: Мир, 1984. - Кн.1. 486 с. - Кн.2. 341 с.

8. Хантер Р. Проектирование и конструирование компиляторов / Р. Хантер. - М.: Финансы и статистика, 1984. - 232 с.

9. Хантер Р. Основные концепции компиляторов / Р. Хантер. - М.: Издательский дом “Вильямс”, 2002. - 256 с.

10. Corcoran W.J. Estimating Reliability after Corrective Action/ W.J. Corcoran, H. Weingarten, P.W. Zehna. Management Science, 10, No 4, p. 786 – 795 (July 1954).

ПРИЛОЖЕНИЕ А
Схема функционирования программы – интерпретатора

ОПИСАНИЕ АЛГОРИТМА

А.1. Наименование и назначение блоков

Под символами Ω и Θ далее понимаются модели памяти и кодов интерпретируемого микропроцессора.

1. Блок задания начальных условий эмуляции: значение счётчика адреса команд РС, содержимого операционных регистров и участков памяти и т.п.

2. Проверка корректности адреса памяти, на который указывает РС. В частности, не вышел ли он за пределы адресного пространства?

3. Операция Ω – преобразования РС, выборка выполняемой команды из Ω – ОЗУ, Ω – преобразование командных адресов операндов в Ω – исполнительные адреса.

4. Проверка, является ли моделируемая команда командой перехода?

5. Выборка, по необходимости, из ОЗУ моделирующей $ПЭВМ_B$ операндов моделируемой команды и их Θ – преобразование.

6. Выполняется ли условие перехода?

7. Выбор необходимой программы интерпретации основной элементарной операции по Θ – коду операции.

8. Подготовка Ω – адреса дальнейшего выполнения интерпретации по операции перехода или прерыванию.

9. Моделирование выполнения операций останова $МП_A$.

10 – 12. Интерпретация одной из основных элементарных операций машинного цикла.

13. Установка или продвижение РС на новый Ω – адрес команды.

14. Анализ результата операции. Результат операции считается нормальным с точки зрения её корректного завершения: отсутствия деления на нуль, конфликтов атрибутов аргументов.

15. Помещение Θ – результатов в Ω – ОЗУ.

16. Инициировано внешнее прерывание?

А.2. Описание логики структурной схемы

Схема, приведённая на рисунке А.1, отражает процесс обработки моделируемой программы так, как будто его выполняет эмулируемый процессор.

Выполнение моделирования инициируется извне через интерфейс программы-интерпретатора путём использования соответствующей функции. При этом обязательно должен быть указан стартовый адрес, с которого начинается эмуляция, и, факультативно, загружены необходимые данные (блок № 1).

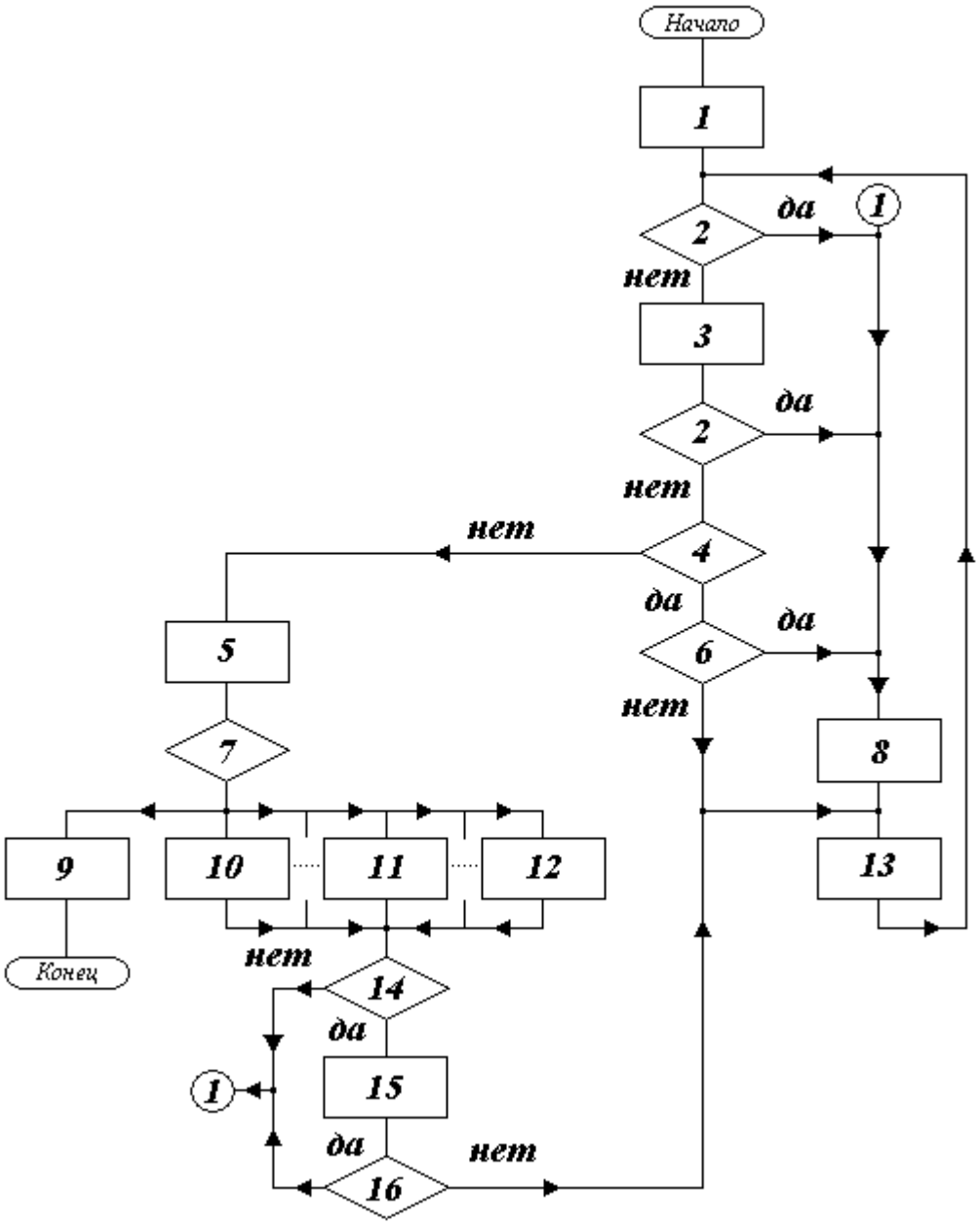

Рисунок А.1 – Обобщённый алгоритм реализации модели машинного цикла

В ходе интерпретации последовательно выполняются неосновные операции машинного цикла, такие как: операция выборки из модели памяти кода операции и адресной части моделируемой команды с построением исполнительных адресов (блок № 3), контролируется их принадлежность адресному пространству микропроцессора (блоки № 2), выборка и загрузка операндов из памяти (блок № 5).

Выбор процедуры, соответствующей линейной основной операции, и запуск её на выполнение происходит в пределах блоков №№ 7, 9 – 12.

Если текущая интерпретируемая команда является командой перехода, вызовом подпрограммы или прерыванием, инициируемым непосредственно

программой, что проверяется в блоке № 4, в этом случае выполняется извлечение или вычисление значение адреса перехода (блок № 8), и инициализация PC этим значением (блок № 13).

После выполнения процедуры, реализующую основную элементарную операцию (арифметическую, логическую или операцию сравнения), необходимо выполнить проверки корректности совершённой операции. Это касается ситуаций деления на ноль, переполнения разрядной сетки при умножении или исчезновение порядка при делении, использование так называемого “мусора” – неопределённых данных и иных особенностей такого рода, что определяется спецификациями отдельных команд конкретного микропроцессора. Разумеется, такие проверки производятся либо до интерпретации команды, либо в её процессе, и вынесение их в отдельный элемент (блок № 14) структурной схемы сугубо условно.

Ненормальное завершение программы моделирования элементарной операции влечёт за собой внутреннее прерывание, инициируемое процессором, и переход на процедуру, его обрабатывающую. В случае “штатного” завершения эмуляции команды, выполняется установка флагов и сохранение результатов по адресам, определённым в ходе исполнения команды (блок № 15).

В ходе выполнения командного цикла процессора могут возникнуть внешние прерывания. Собственно машинный цикл никогда не прерывается, а реакция на ситуацию прерывания осуществляется после его завершения. Наличие прерывания проверяется блоком № 16, а реакция по его обработке конструктивно не отличается от обработки команды перехода.

Завершение программы интерпретации осуществляется в следующих случаях: а) при интерпретации команды останова моделируемого процессора (блок № 9); б) некорректное выполнение какой-либо операции, инициирование которой на моделируемом процессоре приводит к его аварийной остановке (блок № 14); в) поступления прерывания от интерфейса эмулятора (блок № 16).

В остальных случаях программа – эмулятор поддерживает бесконечный цикл, выполнения которого подчиняется интерпретации кодов, находящихся в модели ОЗУ.

Приложение Б.
Примеры интерпретации отдельных элементарных основных и вспомогательных операций микропроцессоров

Б.1. Программная модель основного машинного цикла процессора H6809 [7]

```
PROGRAM H6809 (input, output);
TYPE        byte = ARRAY [7::0] OF bit;
            word=ARRAY [15::0] OF bit;
VAR  MEM : ARRAY [0..65535] OF byte;
       EAR, PC, X, SP    :word;
       IR, A             :byte;
       Z                 :bit;
PROCEDURE Fetch;
BEGIN
       IR : = MEM[PC];  {Read next instruction}
       PC := PC + 1;  {Bump PC to next instruction}
END;
PROCEDURE Execute;
BEGIN
       {Will be defined later}
END;
BEGIN
       PC := 0; {Start at PC=0 on cold start}
       WHILE true DO
                  BEGIN
                        Fetch;
                        Execute;
                  END;
END.
```

Б.2. Процедура моделирования выполнения команд микропроцессора H6809 [7]

```
PROCEDURE Execute;
CONST {Mnemonic-opcode correspondences}
        ADDA#=8BH; ADDA=0BBH; ADDX#=30H; ADDX=31H;
        ANDA#=84H; ANDA=0B4H;
        BEQ=27H; BNE=26H; BRA=20H;
        CLRA=4FH; CMPA#=81H; CMPA=0B1H; CMPX#=8CH; CMPX=0BCH;
        COMA=43H;
        JMP=7EH; JSR=0BDH;
        LDA#=86H; LDA@X=0A6H; LDA=0B6H;
        LDX=0BEH; LDX#=8EH; LDS#=8FH;
        NOP=12H; NEGA=40H;
        RTS=39H;
        STA@X=0A7H; STA=0B7H; STX=0BFH;
VAR opnum: integer;
FUNCTION NextByte : byte;  {Get next byte in instruction stream}
BEGIN
        NextByte := MEM[PC]; PC := PC + 1
END;
PROCEDURE FetchAddr; {Get 2-byte address from instruction stream}
BEGIN
        EAR[15::8] := NextByte; EAR[7::0] := NextByte
END;
PROCEDURE TestByte (b : byte);  {Set Z according to byte value}
BEGIN
        IF b=0 THEN Z:=1 ELSE Z:=0
END;
PROCEDURE TestWord (w : word);  {Set Z according to word value}
BEGIN
        IF w=0 THEN Z:=1 ELSE Z:=0
END;
PROCEDURE Branch; {Sign-extend offset in EAR[7::0] and add to PC}
BEGIN
        IF EAR[7]=0         THEN EAR[15::8] := 0
                            ELSE EAR[15::8] := 0FFH;
        PC := PC + EAR
END;
BEGIN {Statement part of Execute}
        opnum := IR;
        CASE opnum OF
```

```
NOP:
CLRA:     BEGIN A := 0; TestByte(A) END;
COMA:     BEGIN A := Bcom(A); TestByte(A) END;
NEGA:     BEGIN A := -A; TestByte(A) END;
LDA#:     BEGIN A := NextByte; TestByte(A) END;
LDA@X:    BEGIN  A : = MEM [X] ; TestByte (A) END;
LDA:      BEGIN FetchAddr; A := MEM[EAR]; TestByte(A) END;
STA:      BEGIN FetchAddr; MEM[EAR] := A; TestByte(A) END;
STA@X:    BEGIN MEM[X] := A; TestByte(A) END;
ADDA#:    BEGIN A := A + Nextbyte; TestByte(A) END;
ADDA:     BEGIN
          FetchAddr;
          A := A + MEM[EAR];
          TestByte(A)
          END;
ANDA#:    BEGIN A := Band(A,Nextbyte); TestByte(A) END;
ANDA:     BEGIN
                FetchAddr; A := Band(A,MEM[EAR]); TestByte(A)
          END;
CMPA#:    BEGIN TestByte(A-Nextbyte) END;
CMPA:     BEGIN FetchAddr; TestByte(A-MEM[EAR]) END;
LDX#:     BEGIN FetchAddr; X := EAR; TestWord(X) END;
LDX:      BEGIN
          FetchAddr; X[15::8] :=MEM[EAR];
          X[7::0] :=MEM[EAR+1]; TestWord(X)
          END;
STX:      BEGIN
          FetchAddr; MEM[EAR] :=X[15::8];
          MEM[EAR+1] := X[7::0]; TestWord(X)
     END;
CMPX#:    BEGIN FetchAddr; TestWord(X-EAR) END;
CMPX:     BEGIN
          FetchAddr;
          TestWord(X-(MEM[EAR]| MEM[EAR+1]})
          END;
ADDX#:    BEGIN FetchAddr; X := X + EAR; TestWord(X) END;
ADDX:     BEGIN {"|" denotes concatenation of bytes}
          FetchAddr; X := X + (MEM[EAR] | MEM[EAR+1]);
          TestWord(X)
          END;
LDS#:     BEGIN FetchAddr; SP :=EAR; TestWord(SP) END;
BNE:      BEGIN
          EAR[7::0] := NextByte;
```

```
                    IF Z=0 THEN Branch
                    END;
            BEQ:    BEGIN
                    EAR[7::0] := NextByte;
                    IF Z=1 THEN Branch
                    END;
            BRA:    BEGIN EAR[7::0] := NextByte; Branch END;
            JMP:    BEGIN FetchAddr; PC := EAR END;
            JSR:  BEGIN
                    FetchAddr; SP := SP – 2; {Reserve 2 bytes on stack}
                    MEM[SP] := PC[15::8]; {Save PC}
                    MEM[SP+1] := PC[7::0];
                    PC:=EAR; {… and jump to subroutine}
                  END;
            RTS:  BEGIN {Restore PC from top of stack}
                    PC[15::8] := MEM[SP];
                    PC[7::0] := MEM[SP+1];
                    SP := SP + 2;
                  END;
        END; {End Case}
END; {End Execute}
```

Б.3. Моделирование способов адресации, применяемых в ЭВМ PDP-11, на языке Pascal [7]

```
PROGRAM PDP11 (input,output};
CONST PC =  7; SP = 6;
TYPE word = ARRAY [15::0] OF bit;
        byte = ARRAY [7::0] OF bit;
        address = 0..65535;
VAR R:              ARRAY [0..7] OF word; {General registers}
        MEM:        ARRAY [address] OF byte; {Main memory}
        EAR,PSW:                word;
        byteInstr, addrError:       boolean;
FUNCTION RMW(addr : address) : word; {Read a memory word}
BEGIN
        IF addr MOD 2 = 1 THEN BEGIN addrError := true; RMW := 0 END
                                ELSE BEGIN
                                        addrError := false;
                                        RMW[7::0]   := MEM[addr];
                                        RMW[15::8] := MEM[addr+1];
                                        END;
END;
PROCEDURE CalcEAR (opSpec: ARRAY [5::0] OF bit);
VAR mode, reg:          0..7;
BEGIN
        mode  := opSpec[5::3]; reg  :=  opSpec[2::0];
        CASE mode OF
                0: {register, EAR not used}
                1: EAR := R[reg]; {register  indirect}
                2: {auto-increment}
                        BEGIN EAR  :=  R[reg] ;
                        IF byteInstr AND  reg<>PC AND  reg<>SP
                                THEN R[reg] := R[reg] + 1
                                ELSE R[reg]   := R[reg]+2;
                        END;
                3: {auto-increment  indirect}
                        BEGIN
                                EAR := RMW(R[reg]);
                                R[reg]   := R[reg] + 2
                        END;
                4: {auto-decrement}
                        BEGIN
                                IF byteInstr AND  reg<>PC AND reg<>SP
```

```
                                    THEN R[reg] := R[reg] - 1
                                    ELSE R[reg] := R[reg] - 2;
                        EAR := R[reg] ;
                    END;
                5: {auto-decrement  indirect}
                    BEGIN
                        R[reg] :=  R[reg] - 2;
                        EAR := RMW(R[reg])
                    END;
                6: {indexed}
                    BEGIN
                        EAR := RMW(R[PC]);
                        R[PC] :=  R[PC] + 2;
                        EAR := EAR + R[reg]
                    END;
                7: {indexed  indirect}
                    BEGIN
                        EAR :=  RMW(R[PC]);
                        R[PC] :=  R[PC] + 2;
                        EAR :=  RMW(EAR+R[reg]);
                    END;
        END;
END;
{}
END.
```

Б.4. Текст программы моделирования умножения двух 16-тиразрядных чисел без знака [7]

```
PROGRAM UnsignedMultiply (input,output);
CONST wlen =  15;
TYPEword=ARRAY [wlen::0]   OF bit;
VAR cnt : integer;  mpy, mcnd, hiProd, loProd : word;
BEGIN
        read{mpy,mcnd); loProd := 0; hiProd:= 0;     {initialization}
        FOR cnt := 0 TO wlen DO
        BEGIN {Process the multiplier one bit at a time.}
                {If multiplier LSB is  1,  add multiplicand to high-order
                product,  setting BC according to carry from MSB}
                IF mpy[0] = 1  THEN hiProd := Badd(hiProd,mcnd,0)   ELSE BC := 0;
                {Shift hiProd righ, put BC into MSB, and put lost LSB into BC}
                hiProd := Bshr(hiProd,BC);
                {Shift loProd right, picking up LSB that was lost from hiProd}
                loProd:=Bshr(loProd,BC);
                {Now move the next bit of multiplier to the LSB position}
                mpy := Bshr (mpy, 0);
        END ;
writeln(hiProd,loProd);
END.
```

Б.5. Текст программы умножения двух 16-разрядных целых чисел в дополнительном коде [7]

```
PROGRAM SignedMultiply (input,output);
CONST wlen = 15;
TYPE word = ARRAY [wlen::0] OF bit;
VAR cnt : integer; mpy, mcnd, hiProd, loProd : word;
BEGIN
    read(mpy,mcnd); loProd := 0; hiProd := 0;. {lnitialization}
    FOR cnt := 0 TO wlen -1 DO
        BEGIN {Process all mpy bits except MSB.}
            {lf multiplier LSB is 1, add multiplicand to high-order product, set
            BV to 1 on two's-complement overflow.}
            IF mpy[0] = 1       THEN hiProd := Badd(hiProd,mcnd,0)
                                ELSE BV := 0;
            {Shift hiProd right, replicate the correct sign of the result, and put
            the lost LSB into BC}
            hiProd := Bshr(hiProd,Bxor(hiProd[wlen],BV));
            {Shift loProd right, picking up LSB that was lost from hiProd}
            loProd := Bshr(loProd,BC);
            {Move the next bit of mpy to the LSB position}
            mpy := Bshr(mpy,0);
        END;
        {Process the sign bit of mpy. If mpy's LSB is 1, subtract multiplicand
        from high-order product, setting BV to 1 on two's-complement over-
        flow}
        IF mpy[0] = 1       THEN hiProd := Badd(hiProd,Bcom(mcnd),1)
                            ELSE BV := 0;
        {Shift hiProd right, give it the correct sign, and put LSB into BC}
        hiProd := Bshr(hiProd,Bxor(hiProd[wlen],BV));
        {Shift loProd right, picking up the LSB that was lost from hiProd}
        loProd := Bshr(loProd,BC);
        writeln(hiProd,loProd);
    END.
```

Б.6. Текст программы деления двух целых чисел без знака [7]

```
PROGRAM UnsignedDivide   (input,output);
CONST wlen =  15;
TYPE word = ARRAY [wlen::0] OF bit;
VAR         hiDvnd, loDvnd, dvsr, quot, rmdr : word;
            cnt : integer; flag : boolean;
BEGIN
        read (hiDvnd,loDvnd,dvsr) ;
        IF Beq(dvsr,0) THEN writeln('Error, division by 0 not allowed')
        ELSE IF Bls(dvsr,hiDvnd) THEN writeln('Error, quotient too big')
                ELSE
                BEGIN
                        quot := 0;
                        FOR cnt := 0 TO wlen DO
                        BEGIN
                        quot := Bshl(quot,0); {Make room for next quotient bit.}
                        loDvnd := Bshl(loDvnd,0); hiDvnd := Bshl(hiDvnd,BC);
                        {Shift double-word dividend left one bit,old MSB ends up in BC}
                        flag := BC=1; {Set flag if shifted dividend definitely
                                        greater than divisor.}
                        hiDvnd :=Badd(hiDvnd,Bcom(dvsr),1);
                        {Do trial subtraction, set BC if the result is positive}
                        IF flag OR (BC=1) THEN
                                        {Positive result, set quotient bit}
                                        quot[0] := 1
                                        ELSE
                                        {Negative result}
                                        hiDvnd := Badd(hiDvnd,dvsr,0);
                        END;
                        { restore hiDvnd}
                        rmdr := hiDvnd;
                        writeln(quot,rmdr);
                END;
END.
```

Б.7. Текст программы деления целых чисел, представленных в дополнительном коде [7]

```
 PROGRAM SignedDivide (input,output);
 CONST wlen = 15;
 TYPE word = ARRAY [wlen::0] OF bit;
 VAR hiDvnd, loDvnd, dvsr, quot, rmdr : word;
        cnt : integer; negDvsr, negDvnd : boolean;
BEGIN
read (hiDvnd,loDvnd,dvsr);
negDvsr := dvsr[wlen] = 1;  negDvnd := hiDvnd[wlen]=1;
IF negDvsr THEN dvsr := Badd(0,Bcom(dvsr),1); {Make dvsr positive}
IF negDvnd THEN  {Make dvnd positive}
        BEGIN  {A 'double precision' negate}
                loDvnd := Badd(0,Bcom(loDvnd),1);
                hiDvnd := Badd(0,Bcom(hiDvnd),BC);
        END;
IF Beq(dvsr,0) THEN writeln('Error, divide by 0 not allowed')
        ELSE IF Bls(dvsr,hiDvnd) THEN writeln('Error, quotient too big')
                ELSE
                BEGIN {Compute quotient and remainder}
                quot := 0;
                FOR cnt := 0 TO wlen-1 DO
                        BEGIN
                                quot := Bshl(quot,0); {Make room for next quotient bit}
                                {Shift double-word dvnd left one bit.}
                                loDvnd := Bshl(loDvnd,0); hiDvnd := Bshl(hiDvnd,BC);
                                hiDvnd := Badd(hiDvnd,dvsr,1); {Do trial subtraction}
                                IF hiDvnd[wlen]=0 THEN quot[0] := 1;
                                                        {Positive result, set quotient bit}
                                                ELSE hiDvnd := hiDvnd + dvsr;
                                                        {Negative —- restore hiDvnd}
                        END;
                        IF negDvnd THEN rmdr := Badd(0,Bcom(hiDvnd),1)
                                        ELSE rmdr := hiDvnd;
                                        {Remainder gets sign of dividend.}
                        IF (negDvsr AND (NOT negDvnd)) OR
                        ((NOT negDvsr) AND negDvnd)
                        THEN quot := Badd(0,Bcom(quot),1);
                        writeln (quot, rmdr) ;
                END;
END.
```

Заказ № _____ от “ _____ “ _____ 200 _____ . Тираж _____ экз.
Изд-во СевНТУ