Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Промышленная электроника»

Д. А. Литвинов

АППАРАТНО-ПРОГРАММНЫЕ СРЕДСТВА ВЫЧИСЛИТЕЛЬНЫХ МАШИН И СИСТЕМ

ПРАКТИКУМ
по выполнению лабораторных работ
по дисциплине «Вычислительные машины
и системы» для студентов специальности 1-53 07 01
«Промышленная электроника»
дневной формы обучения

Электронный аналог печатного издания

Гомель 2019

УДК 004.2(075.8)
ББК 32.971.3я73
Л64

Рекомендовано к изданию научно-методическим советом
факультета автоматизированных систем ГГТУ им. П. О. Сухого
(*протокол № 10 от 24.05.2017 г.*)

Рецензент: зав. каф. «Информационные технологии»
ГГТУ им. П. О. Сухого канд. техн. наук, доц. *К. С. Курочка*

Литвинов, Д. А.

Л64 Аппаратно-программные средства вычислительных машин и систем : практикум по выполнению лаборатор. работ по дисциплине «Вычислительные машины и системы» для студентов специальности 1-53 07 01 «Промышленная электроника» днев. формы обучения / Д. А. Литвинов. – Гомель : ГГТУ им. П. О. Сухого, 2019. – 44 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

ISBN 978-985-535-408-7.

Рассмотрены программные средства создания сценариев для операционной системы Windows. Предложены практические задания для изучения языка командных файлов, сервера сценариев Windows Script Host, алгоритмов планирования процессов в операционных системах. Рассмотрены элементы архитектуры микропроцессоров – динамическое предсказание переходов и базовые алгоритмы кэширования оперативной памяти.

Для студентов специальности 1–53 07 01 «Промышленная электроника» дневной формы обучения.

УДК 004.2(075.8)
ББК 32.971.3я73

ISBN 978-985-535-408-7

Лабораторная работа № 1

Командный интерфейс операционной системы Windows

Цель работы: изучить интерфейс и язык командного интерпретатора операционной системы Windows.

1.1. Интерфейс командной строки Windows. Командный интерпретатор

В операционной системе Windows, как и в других операционных системах, интерактивные (набираемые с клавиатуры и сразу же выполняемые) команды выполняются с помощью так называемого командного интерпретатора, иначе называемого командным процессором или оболочкой командной строки (command shell). Командный интерпретатор или оболочка командной строки – это программа, которая, находясь в оперативной памяти, считывает набираемые Вами команды и обрабатывает их.

Командная оболочка – весьма мощная среда работы с командами и сценариями. Некоторые команды распознаются и выполняются непосредственно самим командным интерпретатором – такие команды называются внутренними (например, **COPY** или **DIR**), другие команды операционной системы представляют собой отдельные системные программы Windows. Такие команды называются внешними (например, **MORE** или **XCOPY**). Независимо от типа каждая команда должна соответствовать определенным синтаксическим правилам, согласно которым за именем команды идут обязательные или необязательные аргументы. Полный синтаксис команды можно узнать, вызвав справку: **имя команды?**

Полный список команд можно вывести, набрав **HELP** в командной строке. Для запуска командного интерпретатора (открытия нового сеанса командной строки) можно выбрать пункт **Выполнить**… (**Run**) в меню **Пуск** (**Start**) и ввести имя файла **Cmd.exe**. Основные команды командного интерпретатора:

1. **ATTRIB** – отображение и изменение атрибутов файлов.
2. **CALL** – вызов одного пакетного файла из другого.
3. **CD** или **CHDIR** – вывод имени либо смена текущей папки.
4. **CHCP** – вывод либо установка активной кодовой страницы.
5. **CLS** – очистка экрана.
6. **COMP** – сравнение содержимого двух файлов или двух наборов файлов.

7. **COPY** – копирование одного или нескольких файлов.

8. **DATE** – вывод либо установка текущей даты.

9. **DEL** – удаление одного или нескольких файлов.

10. **DIR** – вывод списка файлов и подпапок из указанной папки.

11. **DISKPART** – отображение и настройка свойств раздела диска.

12. **ECHO** – вывод сообщений и переключение режима отображения команд на экране.

13. **ENDLOCAL** – конец локальных изменений среды для пакетного файла.

14. **ERASE** – удаление одного или нескольких файлов.

15. **EXIT** – завершение работы программы CMD.EXE (интерпретатора командных строк).

16. **FC** – сравнение двух файлов или двух наборов файлов и вывод различий между ними.

17. **FIND** – поиск текстовой строки в одном или нескольких файлах.

18. **FINDSTR** – поиск строк в файлах.

19. **FOR** – запуск указанной команды для каждого из файлов в наборе.

20. **FSUTIL** – отображение и настройка свойств файловой системы.

21. **GOTO** – передача управления в отмеченную строку пакетного файла.

22. **GPRESULT** – отображение информации о групповой политике для компьютера или пользователя.

23. **IF** – оператор условного выполнения команд в пакетном файле.

24. **LABEL** – создание, изменение и удаление меток тома для дисков.

25. **NET** – работа с сетевым окружением.

26. **MD**, **MKDIR** – создание папки.

27. **MKLINK** – создание символических и жестких ссылок.

28. **MODE** – конфигурирование системных устройств.

29. **MOVE** – перемещение одного или нескольких файлов из одной папки в другую.

30. **PATH** – отображает или устанавливает путь поиска исполняемых файлов.

31. **PAUSE** – приостанавливает выполнение пакетного файла и выводит сообщение

32. **RD**, **RMDIR** – удаление папки.

33. **REN**, **RENAME** – переименование файла или папки.

34. **ROBOCOPY** – улучшенное средство копирования файлов и деревьев каталогов.

35. **SET** – показывает, устанавливает и удаляет переменные среды Windows.

36. **SETLOCAL** – начинает локализацию изменений среды в пакетном файле.

37. **SC** – отображает и настраивает службы (фоновые процессы).

38. **SCHTASKS** – выполняет команды и запускает программы по расписанию.

39. **SHIFT** – изменение положения (сдвиг) подставляемых параметров для пакетного файла.

40. **SHUTDOWN** – локальное или удаленное выключение компьютера.

41. **START** – выполнение программы или команды в отдельном окне.

42. **TASKLIST** – отображение всех выполняемых задач, включая службы.

43. **TASKKILL** – прекращение или остановка процесса или приложения.

44. **TIME** – вывод и установка системного времени.

45. **XCOPY** – копирование файлов и деревьев каталогов.

Более подробное описание команд приведено в электронном варианте данной лабораторной работы.

1.2. Контрольное задание 1

Выполнить контрольное задание, результат показать преподавателю.

1. Запустить командный процессор: **cmd.exe**

2. Создать на диске E папка: **mkdir test или md test**

3. Войти в созданную папку: **cd test**

4. В папке создать текстовый файл с информацией о студенте (*ФИО*, *дата рождения, группа*): **Copy con my.txt**

Примечание: для окончания ввода нажать CTRL+Z и затем ENTER.

5. Вывести содержимое паки на экран: **dir**

6. Вывести содержимое созданного файла на экран: **type my.txt**

7. В созданной папке создать две папки с именами A1 и A2.

Примечание: для повтора предыдущей команды удобно использовать кнопки – «вверх», «вниз», «вправо».

8. Скопировать *my.txt* в папку A1: **copy my.txt A1**

9. В папке A2 создать файл с деревом каталога *test*: **tree >A2\tree.txt**

10. Переименовать файл в папке A1 в *old.txt*: **ren a1\ my.txt old.txt**

11. С помощью команды *move* поменять содержимое папок A1 и A2.

12. Удалить все папки и файлы в исходной папке *test*: **rd /s**

1.3. Контрольное задание 2

Задание выполняется в командном процессоре. Все команды привести в отчете по пунктам. Количество баллов за выполнение – 1.

1. Создать папку **Zadanie1,** а в ней три папки – A1, A2, A3.

2. В папку A1 скопировать все файлы с расширением **ini** (или другим) из каталога **c:\windows**

3. В папке A2 с помощью команды **dir** создать файл с именами файлов в паке **c:\windows**

4. Скопировать содержимое папок A1 и A2 в A3 и вывести ее содержимое на экран. При выполнении задания использовать группировку команд (**&**)

5. Заменить расширение у всех файлов из папки A1 с **ini** на **bak**

6. Сохранить значения системных переменных окружения в файле с именем **set.txt** в папке A3.

7. Создать свою системную переменную **myname** с фамилией студента. Сохранить значения системных переменных окружения в файле с именем **new_set.txt** в папке A3. Файл привести в отчете.

8. Показать созданную структуру преподавателю. Удалить папки A1, A2, A3.

1.4. Язык оболочки командной строки. Командные файлы

Язык оболочки командной строки (shell language) в Windows реализован в виде командных (или пакетных) файлов. Командный файл в Windows – это текстовый файл с расширением **bat** или **cmd**, в котором записаны команды операционной системы (как внешние, так и внутренние), а также некоторые дополнительные инструкции и ключевые слова, придающие командным файлам некоторое сходство с алгоритмическими языками программирования. Инструкции в сценариях обычно выражаются с использованием правил и синтаксиса соответствующего приложения или служебной утилиты в сочетании с простыми управляющими операторами, такими, как операторы циклов и условные операторы.

1.5. Контрольное задание 3

В соответствии с заданием разработать пакетный файл. Все параметры (исходные данные) передаются через командную строку при запуске командного файла. Выполнить обязательную проверку на наличие всех требуемых параметров. Командный файл и результаты его работы привести в отчете. Обязательна демонстрация работы пакетного файла преподавателю.

Количество и сложность заданий (табл. 1.1) студент определяет самостоятельно. Максимальный возможный бал по работе – 10.

Таблица 1.1

Варианты заданий

Вариант	**Задание**	**Сложность, балл**
1	В заданном месте создать папку с именем зарегистрированного пользователя. В файл с именем «Log In. log» сохранить время и дату входа в систему	3
2	Выполнить резервное копирование файлов заданного расширения из указанной папки (с вложенными) в указанное место. Папки и расширение задать как параметры. Выполнить проверку на наличие заданных параметров, отключить вывод на экран списка копируемых файлов	3
3	Удалить на заданном диске (включая вложенные папки) все файлы с заданным расширением (например, bak, tmp) вне зависимости от его атрибутов. Диск и расширение задать как параметры. Выполнить проверку на наличие заданных параметров	3
4	Скопировать все файлы с заданным расширением в заданную папку без сохранения структуры каталогов (четные номера компьютеров), с сохранением структуры (нечетные номера компьютеров). Папки и расширение задать как параметры	2
5	Сохранить местоположение всех файлов с заданным расширением в файле с указанным именем. Расширение и имя файла задать как параметры. Заархивировать полученный список файлов любым архиватором	3
6	Составить пакетный файл для копирования заданного файла в соответствии со списком папок, хранящихся в другом файле. Имена файлов задать как параметры	3

Окончание табл. 1.1

Вариант	Задание	Сложность, балл
7	Создать на заданном диске папку с именем пользователя, вошедшего в систему, и сделать ее активной. Диск и имя файла задать как параметры	2
8	Создать в указанном месте структуру папок, хранящуюся в заданном файле	3
9	Выполнить резервное копирование с последующей архивацией файлов заданного расширения, находящихся в заданной и вложенных папках	3
10	Разработать пакетный файл для построения системы студенческих каталогов. Исходными данными являются имя группы и число пользователей в группе	3
11	Разработать пакетный файл для построения системы студенческих каталогов. Исходными данными являются имя группы и файл с именами пользовательских папок	4
12	Разработать пакетный файл для копирования только новых и обновленных файлов заданного расширения из одного каталога в другой (включая подкаталоги)	2
13	Разработать пакетный файл для копирования файлов заданного расширения, измененных в указанный день или после, из одного каталога в другой (включая подкаталоги)	2
14	Разработать командный файл для добавления/удаления пользователя в систему. Исходными данными являются: имя пользователя, пароль «добавить/удалить». Дополнительно можно задавать группу для пользователя (+1 балл)	2
15	Разработать командный файл для добавления/удаления пользователей в систему согласно списку. Исходными данными является файл, в котором хранятся имя пользователя, пароль, группа	4

Пример выполнения задания

Выполнить резервное копирование содержимого заданной папки (с вложенными) в указанное место. Папки задать как параметры. Выполнить проверку на наличие заданных параметров.

```
@ECHO OFF
CLS
ECHO Файл %0 копирует каталог %1 в %2
IF -%1==- GOTO NoParam
IF -%2==- GOTO NoParam
XCOPY %1\ %2 /S
GOTO :eof
:NoParam
ECHO Не заданы необходимые параметры командной строки!
PAUSE
```

Лабораторная работа № 2

Сервер сценариев Windows Script Host

Цель работы: изучить объектную модель Windows Script Host. Ознакомиться с технологией создания WSH-сценариев.

2.1. Сценарии Windows Script Host

Windows Script Host (WSH) – это инструмент, предназначенный для создания сценариев, работающих в ОС Windows и поддерживающих технологию **ActiveX Scripting**. В качестве стандартных языков поддерживаются **Visual Basic Script Edition** (**VBScript**) и **JScript**. WSH является удобным инструментом для автоматизации повседневных задач пользователей и администраторов операционной системы Windows. Используя WSH, можно работать с файловой системой компьютера, а также управлять работой других приложений. WSH – сценарий представляют собой текстовые файлы с расширением **js** или **vbs**. Сценарий можно выполнить в двух режимах консольном и графическом. Например:

cscript C:\MyScript.vbs – консольный режим
wscript C:\MyScript.vbs – графический режим

Если сценарий запускается в графическом режиме, то его свойства можно устанавливать с помощью вкладки **Сценарий** (**Script**) диалогового окна, задающего свойства файла в Windows.

2.2. Объектная модель WSH

Объектная модель WSH версии 5.6 состоит из следующих объектов:

1. **WScript** – главный объект, который служит для создания других объектов или связи с ними, содержит сведения о сервере сценариев, а также позволяет вводить данные с клавиатуры и выводить информацию на экран или в окно Windows.
2. **WshArguments** – обеспечивает доступ ко всем параметрам командной строки запущенного сценария или ярлыка Windows.
3. **WshNamed** – обеспечивает доступ к именным параметрам командной строки запущенного сценария.

4. **WshUnnamed** – обеспечивает доступ к безымянным параметрам командной строки запущенного сценария.

5. **WshShell** – позволяет запускать независимые процессы, создавать ярлыки, работать с переменными среды, системным реестром и специальными папками Windows.

6. **WshSpecialFolders** – обеспечивает доступ к специальным папкам Windows.

7. **WshShortcut** – позволяет работать с ярлыками Windows.

8. **WshUrlShortcut** – предназначен для работы с ярлыками сетевых ресурсов.

9. **WshEnvironment** – предназначен для просмотра, изменения и удаления переменных среды.

10. **WshNetwork** – используется при работе с локальной сетью: содержит сетевую информацию для локального компьютера, позволяет подключать сетевые диски и принтеры.

11. **WshScriptExec** – позволяет запускать консольные приложения в качестве дочерних процессов, обеспечивает контроль состояния этих приложений и доступ к их стандартным входным и выходным потокам.

12. **WshController** – позволяет запускать сценарии на удаленных машинах.

13. **WshRemote** – позволяет управлять сценарием, запущенным на удаленной машине.

14. **WshRemoteError** – используется для получения информации об ошибке, возникшей в результате выполнения сценария, запущенного на удаленной машине.

15. **FileSystemObject** – обеспечивающий доступ к файловой системе компьютера.

Более подробное описание команд приведено в электронном варианте данной лабораторной работы.

2.3. Контрольное задание

Создать сценарий, реализующий в консольном или оконном режиме диалог с пользователем в виде меню. Выполнение сценария прекращается выбором пункта меню «Выход». Первый пункт меню должен выводить информацию о создателе (ФИО, группа) и краткое описание выполняемых скриптом действий, остальные пункты реализуют действия, указанные в таблице в соответствии с выбранным ва-

риантом. Все дополнительные параметры сценариев задаются в результате диалога с пользователем. Количество и сложность заданий (табл. 2.1) студент определяет самостоятельно. Максимальный возможный бал по работе – 10.

Таблица 2.1

Варианты заданий

Вариант	**Задание**	**Сложность, балл**
1	Создать папку в указанном месте с заданным именем	3
2	Скопировать заданный файл с указанного места в заданное	3
3	Скопировать все файлы с указанного места в заданное	4
4	Удалить файлы заданного расширения в заданной папке	4
5	Удалить все содержимое заданной папки	3
6	Вывести на экран и сохранить в текстовом файле список папок в указанном месте с датой создания совпадающей с заданной	5
7	Скопировать файлы заданного расширения с указанного места в папку «BackUp» на указанном диске	4
8	Перенести файлы заданного расширения с указанного места в папку «BackUp» на заданном диске	4
9	Вывести на экран и сохранить в текстовом файле полную структуру папок, начиная с заданного места	5
10	Перенести заданный файл с указанного места в заданное	3
11	Переименовать заданную папку	3
12	Архивирование заданной папки	4
13	Архивирование файлов заданного расширения в заданной папке	4
14	Вывести на экран список всех подключенных сетевых ресурсов	4
15	Создать ярлык для вызова заданной программы и разместить его на рабочем столе	4
16	Создать ярлык, ссылающийся на заданную папку, и разместить его на рабочем столе	4
17	Создать ярлык – ссылку на заданный сетевой ресурс и разместить его на рабочем столе	4
18	Вывести на экран путь к заданной специальной папке	4
19	Вывести на экран и сохранить в текстовом файле параметр реестра (выбрать самостоятельно)	5

Окончание табл. 2.1

Вариант	Задание	Сложность, балл
20	Отредактировать заданный параметр реестра (выбрать самостоятельно). Вывести на экран прошлое значение и новое	5
21	Вывести на экран и сохранить в текстовом файле список дисков с их размером	5
22	Вывести на экран и сохранить в текстовом файле список дисков с значением свободного места	5
23	Вывести на экран и сохранить в текстовом файле дату создания и размер заданной папки	4
24	Вывести на экран и сохранить в текстовом файле дату создания и размер заданного файла	4
25	Вывести на экран и сохранить в текстовом файле список папок в заданном каталоге	4
26	Вывести на экран и сохранить в текстовом файле список файлов с заданным расширением в указанном каталоге и подкаталогах.	5
27	Вывести на экран и сохранить в текстовом файле список специальных папок	4
28	Вывести на экран и сохранить в текстовом файле список файлов с заданным расширением в папке «*мои документы*» с датой их создания	5
29	Создать в заданном месте резервную копию ярлыков рабочего стола	5
30	Определить общий размер файлов заданного расширения на указанном диске	5

Пример выполнения задания

Создать сценарий, реализующий в консольном режиме диалог с пользователем в виде меню. Сценарий создает ярлык на заданный сетевой ресурс:

```
'* Имя: Laba2.vbs
'* Язык: VBScript
'* Описание: пример лабораторной работы
'*****************************************************
*
```

```
Dim s
' Выводим строку на экран
do
 WScript.StdOut.WriteLine "МЕНЮ:"
 WScript.StdOut.WriteLine "--------------------------"
 WScript.StdOut.WriteLine "1. Информация об авторе"
 WScript.StdOut.WriteLine "2. Создание ярлыка на заданный
сетевой ресурс"
 WScript.StdOut.WriteLine "3. Выход"
 WScript.StdOut.Write "Выберите пункт меню:"
 ' Считываем строку
 s = WScript.StdIn.ReadLine
 ' Создаем объект WshShell
 Set WshShell = WScript.CreateObject("WScript.Shell")

 if (s="1") Then
 WScript.StdOut.WriteLine "ФИО, группа"

 elseif(s="2") Then
 WScript.StdOut.Write "Введите имя ярлыка:"
 f = WScript.StdIn.ReadLine
 ' Создаем ярлык на сетевой ресурс
 Set oUrlLink = WshShell.CreateShortcut(f+".URL")
 ' Устанавливаем URL
 WScript.StdOut.Write "Введите имя ресурса:"
 f = WScript.StdIn.ReadLine
 oUrlLink.TargetPath = f
 ' Сохраняем ярлык
 oUrlLink.Save
 End if

loop until (s="3")
```

Пример работы с элементами оконного диалога

```
Dim WshShell, Res, Text, Title, vbOkCancel , vbOk, s
' Инициализация констант для диалоговых окон
vbOkCancel = 1
vbOk = 1
```

```
WScript.Echo "Привет!"
s = InputBox("Hello World!", 65, "MsgBox Example")
MsgBox ("You entered: " & s)

' Создание объекта WshShell
set WshShell = WScript.CreateObject("WScript.Shell")
Text = "Вы действительно хотите удалить файлы ?"
Title = "Сообщение"

' Вывод диалогового окна на экран
Res = WshShell.Popup(Text,0, Title, vbOkCancel)

'Определение, какая из кнопок была нажата в диалоговом окне
if (Res = vbOk) then
 WshShell.Popup ("Нажата кнопка ОК")
else
 WshShell.Popup ("Нажата кнопка Отмена")
end if
```

Лабораторная работа № 3

Алгоритмы динамического предсказания переходов

Цель работы: изучить алгоритмы динамического предсказания переходов A1, A2, A3.

3.1. Алгоритмы динамического предсказания переходов

Предсказание переходов на сегодняшний день рассматривается как один из наиболее эффективных способов борьбы с конфликтами по управлению. Идея заключается в том, что еще до момента выполнения команды условного перехода или сразу же после ее поступления на конвейер делается предположение о наиболее вероятном исходе такой команды (переход произойдет или не произойдет). Последующие команды подаются на конвейер в соответствии с предсказанием. Известно более двух десятков различных способов реализации идеи предсказания переходов, отличающихся друг от друга исходной информацией, на основании которой делается прогноз, сложностью реализации и, главное, точностью предсказания. При классификации схем предсказания переходов обычно выделяют два подхода: статический и динамический, в зависимости от того, когда и на базе какой информации делается предсказание.

Идея динамического предсказания переходов предполагает накопление информации об исходе предшествующих команд. История переходов фиксируется в форме таблицы, каждый элемент которой состоит из *m* битов. Общепринятое название таблицы предыстории переходов – таблица истории для шаблонов (PHT, Pattern History Table).

Рис. 3.1. Зависимость точности предсказания от разрядности элементов PHT

При описании динамических схем предсказания переходов их часто расценивают как один из видов автоматов Мура, при этом содержимое элементов РНТ трактуется как информация, отображающая текущее состояние автомата. Существуют различные варианты подобных автоматов, однако реально осмысленно говорить лишь о трех вариантах, которые условно обозначим как A1, A2 и A3.

Автомат A1 имеет только два состояния, поэтому каждый элемент РНТ состоит из одного бита ($m = 1$), значение которого отражает исход последнего выполнения команды условного перехода. Диаграмма состояний автомата A1 приведена на рис. 3.2.

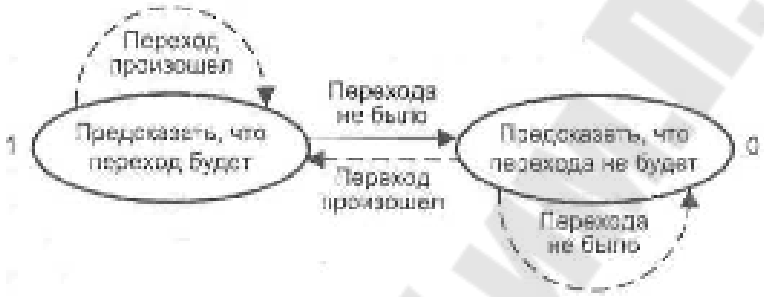


Рис. 3.2. Диаграмма состояний автомата A1

Если команда завершилась переходом, то в соответствующий элемент РНТ заносится единица, иначе – ноль. Очередное предсказание **совпадает с итогом предыдущего** выполнения команды. После обработки очередной команды содержимое элемента корректируется.

Два других автомата предполагают большее число состояний, поэтому в них используются РНТ с многоразрядными элементами. Чаще всего ограничиваются двумя разрядами ($m = 2$) и, соответственно, автоматами с четырьмя состояниями.

Рис. 3.3. Диаграмма состояний автомата A2

В двухразрядном автомате А2 элементы РНТ отражают исходы двух последних выполненных команд условного перехода и заполняются по схеме регистра сдвига. После обработки очередной команды содержимое выделенного этой команде элемента РНТ сдвигается влево на один разряд, а в освободившуюся позицию заносится единица (если переход был) или ноль (если перехода не было). Если в элементе РНТ присутствует хотя бы одна единица, то при очередном выполнении команды делается предсказание, что переход будет. При нулевом значении элемента РНТ считается, что перехода не будет. Диаграмма состояний для такого автомата показана на рис. 3.3.

Рассмотрим работу такого автомата, который будем обозначать А3. При поступлении на конвейер команды условного перехода происходит обращение к соответствующему счетчику, находящемуся в РНТ. В зависимости от текущего состояния счетчика делается прогноз, определяющий дальнейший порядок извлечения команд программы. После прохождения на конвейере ступени исполнения, когда становится известным исход команды, содержимое счетчика увеличивается на единицу, если команда завершилась переходом, или уменьшается на единицу, если перехода не было. Счетчик работает в режиме насыщения. Это значит, что добавление единицы сверх максимального числа в счетчике, а также вычитание единицы при нулевом состоянии счетчика уже не производится, т. е. состояние счетчика в этих случаях остается прежним. Основанием для предсказания служит **старший разряд счетчика**. Если он содержит единицу, то делается предсказание о возможном переходе, в противном случае предполагается, что перехода не будет.

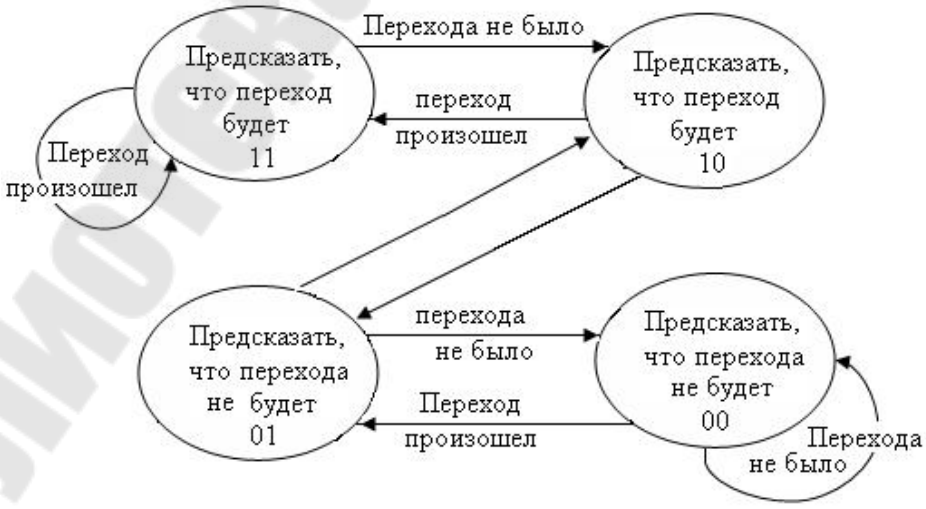


Рис. 3.4. Диаграмма состояний автомата А3

Проведенные исследования показали, что точности предсказания при $m = 2$ и $m = 3$ отличаются незначительно. Поэтому в большинстве известных микропроцессорах используются двухразрядные счетчики ($m = 2$).

3.2. Контрольное задание

Разработать программу, осуществляющую моделирование динамического предсказания переходов на основе алгоритмов A1, A2, A3. История переходов фиксируется в PHT-таблице, состоящей из n элементов. Структура таблицы представлена на рис. 3.5. В первой колонке хранится 16-разрядный адрес команды перехода. Последние две колонки накапливают статистику работы предсказателя.

Адрес команды перехода (16 bit)	**Текущее состояние автомата предсказания**	**Число верно предсказанных переходов**	**Общее число переходов**
0000 h – FFFF h	2	15	22

Рис. 3.5. Структура PHT-таблицы

Алгоритм работы предсказания переходов может быть реализованы с помощью конечных автоматов. В процессе работы программы на экран выводится следующая информация:

1. PHT-таблица или таблицы в зависимости от варианта.
2. Процент успешно предсказанных переходов для каждой команды и общий.
3. Входная информация о работе команд перехода вводится в диалоговом режиме в виде следующей последовательности:

1) адрес команды перехода. При наличии свободных мест в PHT-таблице можно ввести новый адрес, в противном случае выбирается один из существующих адресов;

2) тип команды перехода – **цикл** или **ветвление** (только для новых адресов);

3) если команда перехода – типа **ветвление** случайным образом (генерируется программно) или в диалоговом режиме задается успешный или не успешный переход;

4) Если команда перехода **типа цикл**, случайным образом (генерируется программным способом) или в диалоговом режиме, задается число успешных переходов в диапазоне от 1–20.

Таблица 3.1

Варианты заданий

Вариант	**Число элементов *n* в РНТ-таблице**	**Алгоритм**	**Сложность, балл**
1	4	A1	3
2	6	A2	4
3	8	A3	5
4	10	A2 + A1	6
5	12	A3 + A1	7
6	14	A3 + A2	8

Вариант задания (табл. 3.1) выбирается студентом самостоятельно, оценивая его сложность. Максимальный возможный бал по работе – 10. Дополнительные 2 балла могут быть добавлены преподавателем за:

1) удобный интерфейс с инструментами автоматизации ввода (применение генераторов случайных чисел);

2) использование ассоциативного массива для организации поиска по адресу команды перехода;

3) алгоритмические решения и др.

По согласованию с преподавателем возможно использование других алгоритмов динамического предсказания переходов.

Лабораторная работа № 4

Алгоритмы кэширования данных

Цель работы: изучить организацию кэш-памяти и базовые алгоритмы отображения оперативной памяти в кэш.

4.1. Кэш-память. Кэширование данных

Кэш-память (КП) или кэш представляет собой организованную в виде ассоциативного запоминающего устройства быстродействующую буферную память ограниченного объема, которая располагается между регистрами процессора и относительно медленной основной памятью и хранит наиболее часто используемую информацию совместно с ее признаками (тегами), в качестве которых выступает часть адресного кода.

В процессе работы отдельные блоки информации копируются из основной памяти в кэш-память. При обращении процессора за командой или данными сначала проверяется их наличие в КП. Если необходимая информация находится в кэше, она быстро извлекается. Это **кэш-попадание**. Если необходимая информация в КП отсутствует (**кэш-промах**), то она выбирается из основной памяти, передается в микропроцессор и одновременно заносится в кэш-память.

В структуре кэш-памяти выделяют два типа блоков данных:

– память отображения данных (собственно сами данные, дублированные из оперативной памяти);

– память тегов (признаки, указывающие на расположение кэшированных данных в оперативной памяти).

Пространство памяти отображения данных в кэше разбивается на строки – блоки фиксированной длины (например, 32, 64 или 128 байт). Каждая строка кэша может содержать **непрерывный** выровненный блок байт из оперативной памяти. Какой именно *блок оперативной памяти отображен на данную строку кэша*, *определяется тегом строки* и *алгоритмом отображения*. По алгоритмам отображения оперативной памяти в кэш выделяют три типа кэш-памяти:

– полностью ассоциативный кэш;

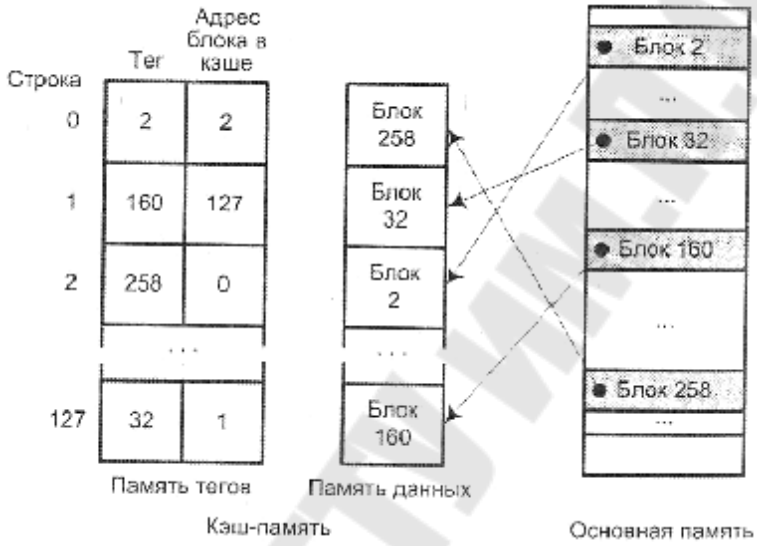
– кэш прямого отображения;

– множественный ассоциативный кэш.

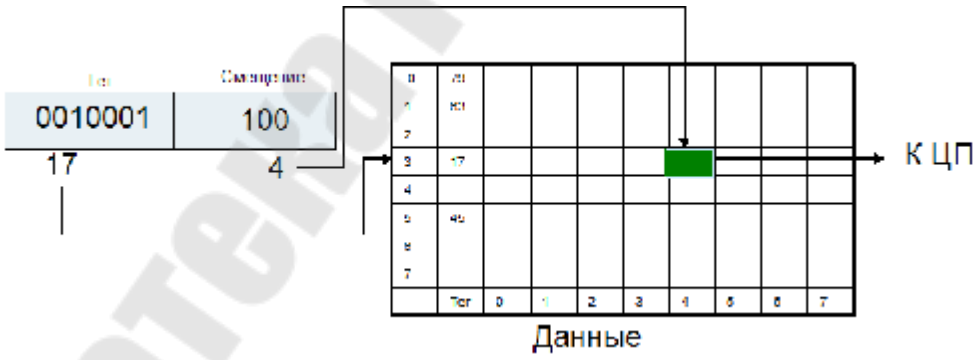
Для **полностью ассоциативного кэша** характерно, что кэш-контроллер может поместить **любой блок оперативной памяти в**

любую строку кэш-памяти (рис. 4.1 *а*). Поле тега совпадает с адресом блока основной памяти. Для проверки наличия копии блока в кэш-памяти логика управления кэша должна **одновременно** проверить теги всех строк на совпадение с полем тега адреса.

Также физический адрес памяти разбивается на две части: **смещение** в блоке (строке кэша) и **номер блока** (рис. 4.1 *б*). При помещении блока в кэш *номер блока сохраняется в теге* соответствующей строки.



а)



б)

Рис. 4.1. Кэш-память с ассоциативным отображением (*а*);
ассоциативный кэш 8x8 для 10-битного адреса (*б*)

Основным достоинством данного способа отображения является хорошая утилизация оперативной памяти, так как нет ограничений на то, какой блок может быть отображен на ту или иную строку кэш-

памяти. К недостаткам следует отнести сложную аппаратную реализацию этого способа, что приводит к увеличению времени доступа к такому кэшу и увеличению его стоимости.

Другой способ отображения оперативной памяти в кэш – это **кэш прямого отображения**. В этом случае адрес памяти (номер блока) однозначно определяет строку кэша, в которую будет помещен данный блок.

Предположим, что ОЗУ состоит из 1000 строк с номерами от 0 до 999, а кэш-память имеет емкость только 100 строк. В кэш-памяти с прямым отображением строки ОЗУ с номерами 0, 100, 200, ..., 900 могут сохраняться только в строке 0 КП и нигде иначе; строки 1, 101, 201, …, 901 и т. д. (рис. 4.2).

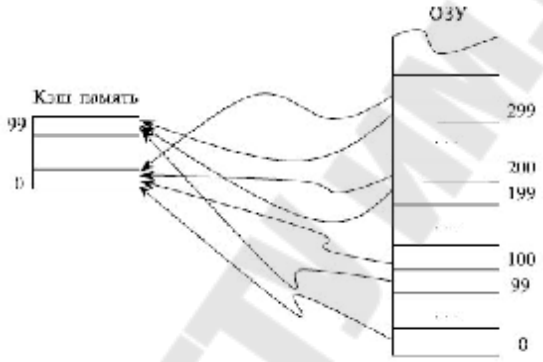


Рис. 4.2. Принцип организации кэш-памяти с прямым отображением

При *прямом отображении* адрес строки i кэш-памяти, на которую может быть отображен блок j из основной памяти, однозначно определяется выражением $i = j \bmod m$, где m – общее число строк в кэш-памяти. Рассмотрим организацию кэш-памяти с прямым отображением, представленную на рис. 4.3.

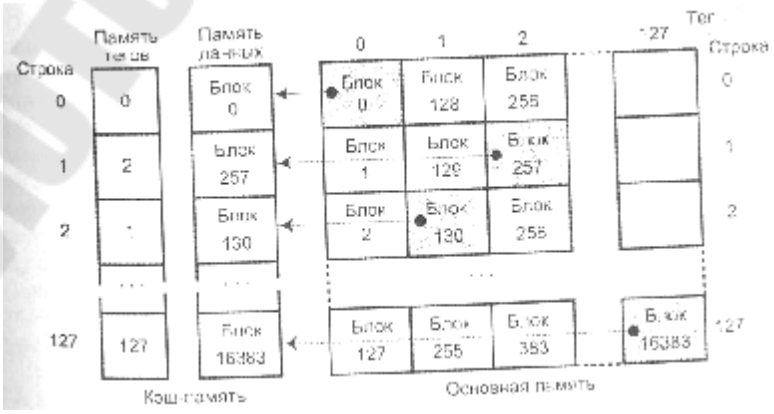


Рис. 4.3. Организация кэш-памяти с прямым отображением

При реализации такого отображения 14-разрядный адрес блока основной памяти ($2^{14} = 16384$) условно разбивается на два поля. Логика кэш-памяти интерпретирует эти 14 бит как 7-разрядный тег и 7-разрядное поле строки. Поле строки указывает на одну из $128 = 2^7$ строк кэш-памяти, а именно на ту, куда может быть отображен блок с заданным адресом (**строка**). **Поле тега** определяет, какой именно из **списка блоков**, закрепленных за данной строкой кэша, будет отображен. Тегом служат *семь старших разрядов* адреса блока.

В нашем примере $i = j \bmod 128$, где i может принимать значения от 0 до 127, а адрес блока j – от 0 до 16383:

Адрес блока	Тег	Строка
130 =	0000001	0000010
257 =	0000010	0000001
16383 =	1111111	1111111

Дополнительно физический адрес разбивается еще и на смещение в блоке (рис. 4.4).

Рис. 4.4. Кэш прямого отображения 8x8 для 10-битного адреса

Преимуществами данного алгоритма являются простота и дешевизна реализации. Основной его недостаток – жесткое закрепление за определенными блоками оперативной памяти одной и той же строки в кэше, что снижает эффективность такого кэша из-за вероятных частых перезагрузок строк.

4.2. Контрольное задание

Разработать программу, реализующую алгоритм отображения кэш-памяти. Вариант задания (табл. 4.1) выбирается студентом самостоятельно, оценивая его сложность. По согласованию с преподавате-

лем возможно использование других алгоритмов кеширования данных. Максимальный возможный балл по работе – 10. Дополнительные 2 балла могут быть добавлены преподавателем за:

1) удобный интерфейс с инструментами автоматизации ввода (применение генераторов случайных чисел);

2) использование оригинальных подходов к хранению данных;

3) алгоритмические решения и др.

Таблица 4.1

Варианты заданий

Вариант	**Алгоритм кэширования данных**	**Сложность, балл**
1A	Полностью ассоциативный кэш. Однокомпонентная адресация. Значение тега совпадает с адресом ячейки в основной памяти. Кэширование данных из основной памяти осуществляется по одной ячейке	4
1B	Полностью ассоциативный кэш. Двухкомпонентная адресация. Размещение данных в кэш-памяти определяется тегом и смещением. Размер строки – 4 ячейки. Кэширование данных из основной памяти осуществляется по одной ячейке	5
1C	Полностью ассоциативный кэш. Двухкомпонентная адресация. Размещение данных в кэш-памяти определяется тегом и смещением. Размер строки – 4 ячейки. Кэширование данных из основной памяти осуществляется блоком. Величина блока – 4 ячейки	6
2A	Кэш прямого отображения. Двухкомпонентная адресация. Размещение данных в кэш-памяти определяется тегом и строкой. Кэширование данных из основной памяти осуществляется по одной ячейке	5
2B	Кэш прямого отображения. Трехкомпонентная адресация. Размещение данных в кэш памяти определяется тегом, строкой и смещением. Размер строки – 4 ячейки. Кэширование данных из основной памяти осуществляется по одной ячейке	7
2C	Кэш прямого отображения. Трехкомпонентная адресация. Размещение данных в кэш-памяти определяется тегом, строкой и смещением. Размер строки – 4 ячейки. Кэширование данных из основной памяти осуществляется блоком. Величина блока – 4 ячейки	8

В работе необходимо организовать кэширование сегмента основной памяти размером 0xFF – 256 ячеек в соответствии с выбран-

ным вариантом. Кэш-память заполняется последовательно, при полном заполнении замещение данных начинается с начала. Программа должна накапливать общую статистику «попаданий» в кэш и число обращений к основной памяти. Моделирование кэширования выполнить с общими размерами области данных кэш 8, 16 и 32 ячеек, сравнить полученные результаты. При многокомпонентной адресации необходимо учитывать, что число строк памяти зависит от размера строки. Число обращений к кэш-памяти – не менее двухкратного размера кэш.

Работа программы начинается с заполнения сегмента основной памяти случайными числами и задания размера кэш-памяти (не более 32 ячеек для хранения данных). Размер кэш кратен степени 2. Далее циклически реализуется следующая последовательность действий:

1. Запрос у пользователя адреса ячейки памяти, к которой происходит обращение.

2. Определить значение тега для запрашиваемой ячейки. Проверить, есть ли такой в кэш-памяти.

3. Если фиксируется «попадание», то оно накапливается в статистике и на кран выводится соответствующая запись в кэш-памяти и значение данных в запрашиваемой ячейке.

4. При «промахе» данные из сегмента основной памяти записываются в кэш в соответствии с алгоритмом, определяемым вариантом. В вариантах A, B в кэш переносится одна ячейка из основной памяти, в варианте C – в кэш переносится блок данных их основной памяти.

Программа должна позволять в любой момент времени вывести на экран содержимое всей кэш-памяти вместе с тегами. В программе для описания строки кэш-памяти рекомендуется использовать структуру данных.

4.3. Примеры организации кэш-памяти с ассоциативным отображением

В полностью ассоциативном кэше и однокомпонентной модели памяти тега его значение равно адресу ячейки сегмента основной памяти.

В полностью ассоциативном кэше и двухкомпонентной модели памяти тега размещение данных в кэш памяти определяется тегом и смещением в строке. Для строки состоящей из 4 ячеек памяти:

Адрес = | Тег | Смещение | = | Тег | 0 | 1 | 2 | 3 |

Так, для ячейки сегмента основной памяти с адресом 0xA6 и строки кэш на 4 ячейки данных тег и смещение будут:

Адрес = 0xA6 = 166	Тег (166/4 = 41)	Смещение (166%4 = 2)
0b10100110	0b101001 = 41	0b10 = 2

При блочном кэшировании данных адрес начала блока, переносимого из основной памяти, можно определить как:

Адрес блока = Тег × (Величина строки кэш) + 1 т. е. 41 × 4 + 1 = 165.

При двухкомпонентной адресации необходимо учитывать, что число строк в моделируемой кэш-памяти зависит от размера строки. Так, если по заданию общий размер области данных кэш-памяти составляет 8 ячеек, а длина строки – 4 ячейки, то итоговое число строк 8/4 = 2.

4.4. Примеры организации кэш-памяти с прямым отображением

В кэш прямого отображения и двухкомпонентной модели памяти тега размещение данных в кэш памяти определяется тегом и строкой:

Адрес = | Тег | Строка |

Значение тега определяется размером кэш-памяти (максимальное число строк в кэш). Так, для ячейки сегмента основной памяти с адресом 0xA6 и размере кэш памяти – 16 строк, тег и строка будут:

Адрес = 0xA6 = 166	Тег (166/16 = 10)	Строка (166%16 = 6)
0b10100110	0b1010 = 10	0b0110 = 6

В кэш прямого отображения и трехкомпонентной модели памяти тега размещение данных в кэш-памяти определяется тегом и строкой. Для строки, состоящей из 4 ячеек памяти:

Адрес = | Тег | Строка | Смещение | = | Тег | Строка | 0 | 1 | 2 | 3 |

Так, для ячейки сегмента основной памяти с адресом 0xA6, размере кэш-памяти – 16 строк и строки кэш на 4 ячейки данных тег, строка и смещение будут:

Адрес = 0xA6 = 166 0b10100110	Тег (166/16 = 10) 0b1010 = 10	Строка (166%16)/4 = 1 0b01 = 1	Смещение (166%16)%4 = 2 0b10 = 2

При трехкомпонентной адресации необходимо учитывать, что число строк в моделируемой кэш-памяти зависит от размера строки. Так, если по заданию общий размер области данных кэш-памяти составляет 8 ячеек, а длина строки – 4 ячейки, то итоговое число строк 8/4 = 2.

Лабораторная работа № 5

Алгоритмы планирования процессов

Цель работы: изучить алгоритмы планирования процессов.

5.1. Алгоритмы кратковременного планирования процессов

Для любого процесса, находящегося в вычислительной системе, вся информация, необходимая для совершения операций над ним, доступна операционной системе. Обычно она хранится в одной или нескольких структурах (таблицах) данных, которые принято называть PCB (Process Control Block) или блоком управления процессом. Блок управления процессом является моделью процесса для операционной системы. Любая операция, производимая операционной системой над процессом, вызывает определенные изменения в PCB. В рамках принятой модели состояний процессов содержимое PCB между операциями остается постоянным.

Существует большой набор разнообразных алгоритмов планирования, которые предназначены для достижения различных целей и эффективны для разных классов задач. Многие из них могут использоваться на нескольких уровнях планирования. Рассмотрим наиболее распространенные алгоритмы.

5.2. First-Come, First-Served (FCFS)

Простейшим алгоритмом планирования является алгоритм, который принято обозначать аббревиатурой **FCFS** по первым буквам его английского названия – **First-Come, First-Served** («первым пришел, первым обслужен»). Пусть процессы, находящиеся в состоянии *готовность*, выстроены в очередь. Когда процесс переходит в состояние *готовность*, он (ссылка на его PCB) помещается в конец этой очереди. Выбор нового процесса для исполнения осуществляется из начала очереди с удалением оттуда ссылки на его PCB. Очередь подобного типа имеет в программировании специальное наименование – FIFO), сокращение от First In, First Out («первым вошел, первым вышел»).

Такой алгоритм выбора процесса осуществляет невытесняющее планирование. Процесс, получивший в свое распоряжение процессор,

занимает его до истечения текущего CPU burst. После этого для выполнения выбирается новый процесс из начала очереди. Преимуществом алгоритма FCFS является легкость его реализации, но в то же время он имеет и недостатки.

Таблица 5.1

Данные о планируемых процессах

Процесс	p0	p1	p2
Продолжительность очередного CPU burst	13	4	1

Рассмотрим пример. Пусть в состоянии *готовность* находятся три процесса p0, p1 и p2, для которых известны времена их очередных CPU burst. Эти времена приведены в табл. 5.1 в некоторых условных единицах. Для определенности будем полагать, что деятельность процессов ограничивается использованием только одного промежутка CPU burst, что процессы не совершают операций ввода-вывода и что время переключения контекста так мало, что им можно пренебречь.

Если процессы расположены в очереди процессов, готовых к исполнению, в порядке p0, p1, p2, то последовательность их выполнения представлена на рис. 5.1. Первым для выполнения выбирается процесс p0, который получает процессор на все время своего CPU burst, т. е. на 13 единиц времени. После его окончания в состояние *исполнение* переводится процесс p1, он занимает процессор на 4 единицы времени. И, наконец, возможность работать получает процесс p2. Время ожидания для процесса p0 составляет 0 единиц времени; для процесса p1 – 13 единиц; для процесса p2 – 13 + 4 = 17 единиц. Таким образом, среднее время ожидания в этом случае – (0 + 13 + 17)/3 = 10 единиц времени. Полное время выполнения для процесса p0 составляет 13 единиц времени, для процесса p1 – 13 + 4 = 17 единиц, для процесса p2 – 13 + 4 + 1 = 18 единиц. Среднее полное время выполнения оказывается равным (13 + 17 + 18)/3 = 16 единицам времени.

Рис. 5.1. Алгоритм FCFS. Выполнение процессов в порядке p0, p1, p2

Если те же самые процессы расположены в порядке p2, p1, p0, то последовательность их выполнения будет соответствовать рис. 5.2. Время ожидания для процесса p0 равняется 5 единицам времени; для процесса p1 – 1 единице; для процесса p2 – 0 единиц. Среднее время ожидания составит (5 + 1 + 0)/3 = 2 единицы времени. Это в 5 (!) раз меньше, чем в предыдущем случае. Полное время выполнения для процесса p0 получается равным 18 единицам времени; для процесса p1 – 5 единицам; для процесса p2 – 1 единице. Среднее полное время выполнения составляет (18 + 5 + 1)/3 = 8 единиц времени, что почти в 2 раза меньше, чем при первой расстановке процессов.

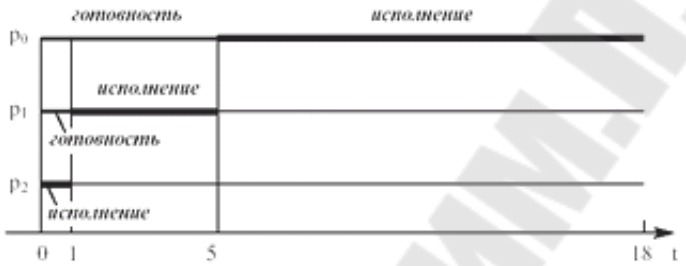


Рис. 5.2. Алгоритм FCFS. Выполнение процессов в порядке p2, p1, p0

Как видно, среднее время ожидания и среднее полное время выполнения для этого алгоритма существенно зависят от порядка расположения процессов в очереди. Если есть процесс с длительным CPU burst, то короткие процессы, перешедшие в состояние *готовность* после длительного процесса, будут долго ждать начала выполнения. Поэтому алгоритм FCFS практически неприменим для систем разделения времени – слишком большим получается среднее время отклика в интерактивных процессах.

5.3. Round Robin (RR)

Модификацией алгоритма FCFS является алгоритм, получивший название **Round Robin** (RR). По сути, это тот же алгоритм, только реализованный в режиме вытесняющего планирования. Пусть все готовые к выполнению процессы организуют циклическую очередь, в которой каждый процесс получает в свое распоряжение процессор на строго фиксированный небольшой квант времени, обычно 10–100 мс (рис. 5.3.). В это время процесс находится в состоянии *исполнение*.

Рис. 5.3. Алгоритм Round Robin

Реализуется такой алгоритм так же, как и предыдущий, с помощью организации процессов, находящихся в состоянии *готовность*, в очередь FIFO. Планировщик выбирает для очередного исполнения процесс, расположенный в начале очереди, и устанавливает таймер для генерации прерывания по истечении определенного кванта времени. При выполнении процесса возможны два варианта:

1. Время непрерывного использования процессора, необходимое процессу (остаток текущего CPU burst), меньше или равно продолжительности кванта времени. Тогда процесс **сам освобождает процессор** до истечения кванта времени, на исполнение поступает новый процесс из начала очереди, и таймер начинает отсчет кванта заново.

2. Продолжительность остатка текущего CPU burst процесса больше, чем квант времени. Тогда по истечении этого кванта **процесс прерывается** таймером и помещается в конец очереди процессов, готовых к исполнению, а процессор выделяется для использования процессу, находящемуся в ее начале.

Рассмотрим предыдущий пример с порядком процессов p0, p1, p2 и величиной кванта времени, равной 4. Выполнение этих процессов иллюстрируется табл. 5.2. Обозначение «И» – для процесса, находящегося в состоянии исполнение, обозначение «Г» – для процессов в состоянии *готовность*, пустые ячейки соответствуют завершившимся процессам.

Таблица 5.2

Последовательность выполнения процессов

Время	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
p0	И	И	И	И	Г	Г	Г	Г	Г	И	И	И	И	И	И	И	И	И
p1	Г	Г	Г	Г	И	И	И	И										
p2	Г	Г	Г	Г	Г	Г	Г	Г	И									

Первым для исполнения выбирается процесс p0. Продолжительность его CPU burst больше, чем величина кванта времени, и поэтому процесс исполняется до истечения кванта, т. е. в течение 4 единиц времени. После этого он помещается в конец очереди готовых к исполнению процессов, которая принимает вид p1, p2, p0. Следующим начинает выполняться процесс p1. Время его исполнения совпадает с величиной выделенного кванта, поэтому процесс работает до своего завершения. Теперь очередь процессов в состоянии *готовность* состоит из двух процессов – p2 и p0. Процессор выделяется процессу p2. Он завершается до истечения отпущенного ему процессорного времени, и очередные кванты отдаются процессу p0 как единственному не закончившему к этому моменту свою работу. Время ожидания для процесса p0 (количество символов «Г» в соответствующей строке) составляет 5 единиц времени, для процесса p1 – 4 единицы времени, для процесса p2 – 8 единиц времени. Таким образом, среднее время ожидания для этого алгоритма получается равным (5 + 4 + 8)/3 = 5,6(6) единицы времени. Полное время выполнения для процесса p0 (количество непустых столбцов в соответствующей строке) составляет 18 единиц времени; для процесса p1 – 8 единиц; для процесса p2 – 9 единиц. Среднее полное время выполнения оказывается равным (18 + 8 + + 9)/3 = 11,6 единицы времени.

На производительность алгоритма RR сильно влияет величина кванта времени. Рассмотрим тот же самый пример для величины кванта времени, равной 1 (табл. 5.3). Время ожидания для процесса p0 составит 5 единиц времени; для процесса p1 – тоже 5 единиц; для процесса p2 – 2 единицы. В этом случае среднее время ожидания получается равным (5 + 5 + 2)/3 = 4 единицам времени. Среднее полное время исполнения составит (18 + 9 + 3)/3 = 10 единиц времени.

Таблица 5.3

Последовательность выполнения процессов

Время	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
p0	И	Г	Г	И	Г	И	Г	И	Г	И	И	И	И	И	И	И	И	И
p1	Г	И	Г	Г	И	Г	И	Г	И									
p2	Г	Г	И															

При очень больших величинах кванта времени, когда каждый процесс успевает завершить свой CPU burst до возникновения прерывания по времени, алгоритм RR вырождается в алгоритм FCFS. При очень малых величинах создается иллюзия того, что каждый из n процессов работает на собственном виртуальном процессоре с производительностью ~ $1/n$ от производительности реального процессора. В реальных условиях при слишком малой величине кванта времени и, соответственно, слишком частом переключении контекста накладные расходы на переключение резко снижают производительность системы.

5.4. Shortest-Job-First (SJF)

При рассмотрении алгоритмов FCFS и RR мы выяснили, что существенным является порядок расположения процессов в очереди. Если короткие задачи расположены в очереди ближе к ее началу, то общая производительность этих алгоритмов значительно возрастает. Если известно время следующих CPU burst для процессов, находящихся в состоянии готовность, то можно выбрать для исполнения процесс не из начала очереди, а процесс с минимальной длительностью CPU burst. Если же таких процессов два или больше, то для выбора одного из них можно использовать уже известный нам алгоритм FCFS. Квантование времени при этом не применяется. Описанный алгоритм получил название «кратчайшая работа первой» или **Shortest Job First** (SJF).

SJF – алгоритм краткосрочного планирования: может быть как вытесняющим, так и невытесняющим. При невытесняющем SJF-планировании процессор предоставляется избранному процессу на все необходимое ему время независимо от событий, происходящих в вычислительной системе. При вытесняющем SJF-планировании учитывается появление новых процессов в очереди, готовых к исполнению во время работы выбранного процесса. Если CPU burst нового

процесса меньше, чем остаток CPU burst у исполняющегося, то исполняющийся процесс вытесняется новым.

Рассмотрим пример работы невытесняющего алгоритма SJF. Пусть в состоянии готовность находятся четыре процесса, p0, p1, p2 и p3, для которых известны времена их очередных CPU burst. Эти времена приведены в табл. 5.4.

Таблица 5.4

Данные о планируемых процессах

Процесс	p0	p1	p2	p3
Продолжительность очередного CPU burst	5	3	7	1

При использовании невытесняющего алгоритма SJF первым для исполнения будет выбран процесс p3, имеющий наименьшее значение продолжительности очередного CPU burst. После его завершения для исполнения выбирается процесс p1, затем p0 и, наконец, p2. Эта картина отражена в табл. 5.5.

Таблица 5.5

Последовательность выполнения процессов

Время	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
p0	Г	Г	Г	Г	И	И	И	И	И							
p1	Г	И	И	И												
p2	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И
p3	И															

Среднее время ожидания для алгоритма SJF составляет (4 + 1 + + 9 + 0)/4 = 3,5 единицы времени. Для алгоритма FCFS при порядке процессов p0, p1, p2, p3 эта величина будет равняться (0 + 5 + 8 + + 15)/4 = 7 единицам времени, т. е. в два раза больше, чем для алгоритма SJF.

Для рассмотрения примера вытесняющего SJF планирования мы возьмем ряд процессов p0, p1, p2 и p3 с различными временами CPU burst и различными моментами их появления в очереди процессов, готовых к исполнению (табл. 5.6).

Таблица 5.6

Данные о планируемых процессах

Процесс	Время появления в очереди очередного CPU burst	Продолжительность
p0	0	6
p1	2	2
p2	6	7
p3	0	5

В начальный момент времени в состоянии *готовность* находятся только два процесса, p0 и p3. Меньшее время очередного CPU burst оказывается у процесса p3, поэтому он и выбирается для исполнения (табл. 5.7). По прошествии 2 единиц времени в систему поступает процесс p1. Время его CPU burst меньше, чем остаток CPU burst у процесса p3, который вытесняется из состояния *исполнение* и переводится в состояние *готовность*. По прошествии еще 2 единиц времени процесс p1 завершается, и для исполнения вновь выбирается процесс p3. В момент времени $t = 6$ в очереди процессов, готовых к исполнению, появляется процесс p2, но поскольку ему для работы нужно 7 единиц времени, а процессу p3 осталось трудиться всего 1 единицу времени, то процесс p3 остается в состоянии *исполнение*. После его завершения в момент времени $t = 7$ в очереди находятся процессы p0 и p2, из которых выбирается процесс p0. Последним получит возможность выполняться процесс p2.

Таблица 5.7

Последовательность выполнения процессов

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
p0	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И							
p1			И	И																
p2							Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И
p3	И	И	Г	Г	И	И	И													

Основную сложность при реализации алгоритма SJF представляет невозможность точного знания продолжительности очередного CPU burst для исполняющихся процессов. При краткосрочном планировании мы можем делать только прогноз длительности следующего CPU burst, исходя из предыстории работы процесса.

5.5. Приоритетное планирование

При приоритетном планировании каждому процессу присваивается определенное числовое значение – приоритет, в соответствии с которым ему выделяется процессор. Процессы с одинаковыми приоритетами планируются в порядке FCFS. Для алгоритма SJF в качестве такого приоритета выступает оценка продолжительности следующего CPU burst. Чем меньше значение этой оценки, тем более высокий приоритет имеет процесс.

Алгоритмы назначения приоритетов процессов могут опираться как на внутренние параметры, связанные с происходящим внутри вычислительной системы, так и на внешние по отношению к ней. К внутренним параметрам относятся различные количественные и качественные характеристики процесса, такие, как ограничения по времени использования процессора, требования к размеру памяти, число открытых файлов и используемых устройств ввода-вывода, отношение средних продолжительностей I/O burst к CPU burst и т. д. Алгоритмы SJF и гарантированного планирования используют внутренние параметры. В качестве внешних параметров может выступать важность процесса. Планирование с использованием приоритетов может быть как вытесняющим, так и невытесняющим. При вытесняющем планировании процесс с более высоким приоритетом вытесняет исполняющийся процесс с более низким приоритетом. В случае невытесняющего планирования он просто становится в начало очереди готовых процессов.

Пусть в очередь процессов, находящихся в состоянии *готовность*, поступают те же процессы, что и в примере для вытесняющего алгоритма SJF, только им дополнительно присвоены приоритеты (табл. 5.8). Будем предполагать, что большее значение соответствует меньшему приоритету, т. е. наиболее приоритетным в нашем примере является процесс p3, а наименее приоритетным – процесс p0.

Таблица 5.8

Данные о планируемых процессах

Процесс	Время появления в очереди	Продолжительность очередного CPU burst	Приоритет
p0	0	6	4
p1	2	2	3
p2	6	7	2
p3	0	5	1

При невытесняющем приоритетном планировании первым для выполнения в момент времени $t = 0$ выбирается процесс p3, как обладающий наивысшим приоритетом. После его завершения в момент времени $t = 5$ в очереди процессов, готовых к исполнению, окажутся два процесса p0 и p1. Большим приоритетом обладает p1, он и начнет выполняться (табл. 5.9). Затем в момент времени $t = 8$ для исполнения будет избран процесс p2, и лишь потом – процесс p0.

Таблица 5.9

Последовательность выполнения процессов

Время	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
p0	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И
p1			Г	Г	Г	И	И													
p2							Г	И	И	И	И	И	И	И						
p3	И	И	И	И	И															

В случае вытесняющего приоритетного планирования (табл. 5.10) первым, как и в предыдущем случае, начнет исполняться процесс p3, а по его окончании – процесс p1. Однако в момент времени $t = 6$ он будет вытеснен процессом p2 и продолжит свое выполнение только в момент времени $t = 13$. Последним, как и раньше, будет исполняться процесс p0.

В рассмотренном выше примере приоритеты процессов с течением времени не изменялись. Такие приоритеты принято называть статическими. Более гибкими являются динамические приоритеты процессов, изменяющие свои значения по ходу исполнения процессов. Схемы с динамической приоритетностью гораздо сложнее в реализации и связаны с большими издержками по сравнению со статическими схемами. Однако их использование предполагает, что эти издержки оправдываются улучшением работы системы.

Таблица 5.10

Последовательность выполнения процессов

Время	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
p0	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И
p1			Г	Г	Г	И	Г	Г	Г	Г	Г	Г	Г	И						
p2							И	И	И	И	И	И	И							
p3	И	И	И	И	И															

5.6. Контрольное задание 1. Невытесняющие алгоритмы планирования процессов

В соответствии с вариантом (**номер по журналу**) для данных, приведенных в табл. 5.11, выполнить следующие алгоритмы планирований: **First-Come, First-Served** прямой и обратный (1 балл), **Round Robin** (1 балл), **Shortest-Job-First** невытесняющий (2 балла), **Shortest-Job-First** невытесняющий, приоритетный (2 балла).

Вычислить полное время выполнения всех процессов и каждого в отдельности, время ожидания для каждого процесса. Рассчитать среднее время выполнения процесса и среднее время ожидания. Результаты оформить в виде таблиц, иллюстрирующих работу процессов.

5.7. Контрольное задание 2. Вытесняющие алгоритмы планирования процессов

В соответствии с вариантом (**номер по журналу**) для данных, приведенных в табл. 5.11, выполнить вытесняющие алгоритмы планирования – **Shortest-Job-First** (2 балла) и **Shortest-Job-First** приоритетный (2 балла).

Вычислить полное время выполнения все процессов и каждого в отдельности, время ожидания для каждого процесса. Рассчитать среднее время выполнения процесса и среднее время ожидания. Результаты оформить в виде таблиц, иллюстрирующих работу процессов.

Сравнить полученные результаты и сделать выводы о эффективности рассмотренных алгоритмов.

Таблица 5.11

Варианты заданий

Вариант	Продолжительности процессов	Время появления в очереди	Приоритеты процессов
1	P0 – 5; P1 – 8; P2 – 2; P3 – 4	P0 – 0; P1 – 3; P2 – 1; P3 – 4	P0 – 2; P1 – 1; P2 – 3; P3 – 3
2	P0 – 10; P1 – 3; P2 – 3; P3 – 7	P0 – 2; P1 –0 ; P2 – 4; P3 –5	P0 – 3; P1 – 2; P2 – 3; P3 – 1
3	P0 – 5; P1 – 8; P2 – 2; P3 – 4	P0 – 3; P1 – 1; P2 – 3; P3 –0	P0 – 3; P1 – 2; P2 – 3; P3 – 1
4	P0 – 2; P1 – 5; P2 – 9; P3 – 3	P0 – 0; P1 – 4; P2 – 6; P3 – 8	P0 – 1; P1 – 3; P2 – 3; P3 – 2
5	P0 – 7; P1 – 8; P2 – 3; P3 – 4	P0 – 4; P1 – 0; P2 – 3; P3 – 5	P0 – 1; P1 – 2; P2 – 3; P3 – 4
6	P0 – 9; P1 – 2; P2 – 2; P3 – 3	P0 – 3; P1 – 0; P2 – 0; P3 – 4	P0 – 3; P1 – 1; P2 – 3; P3 – 3
7	P0 – 7; P1 – 2; P2 – 3; P3 – 1	P0 – 0; P1 – 4; P2 – 3; P3 – 0	P0 – 1; P1 – 3; P2 – 2; P3 – 3
8	P0 – 2; P1 – 3; P2 – 1; P3 – 8	P0 – 4; P1 – 4; P2 – 0; P3 – 0	P0 – 3; P1 – 1; P2 – 3; P3 – 4
9	P0 – 3; P1 – 5; P2 – 2; P3 – 7	P0 – 3; P1 – 0; P2 – 1; P3 – 7	P0 – 1; P1 – 3; P2 – 4; P3 – 1

Окончание табл. 5.11

Вариант	Продолжительности процессов	Время появления в очереди	Приоритеты процессов
10	P0 – 7; P1 – 3; P2 – 4; P3 – 6	P0 – 1; P1 – 2; P2 – 4; P3 – 0	P0 – 2; P1 – 1; P2 – 3; P3 – 4
11	P0 – 3; P1 – 2; P2 – 5; P3 – 6	P0 – 3; P1 – 4; P2 – 4; P3 – 0	P0 – 1; P1 – 1; P2 – 3; P3 – 3
12	P0 – 8; P1 – 3; P2 – 10; P3 – 2	P0 – 8; P1 – 4; P2 – 0; P3 – 3	P0 – 2; P1 – 2; P2 – 1; P3 – 4
13	P0 – 3; P1 – 7; P2 – 1; P3 – 3	P0 – 4; P1 – 0; P2 – 0; P3 – 0	P0 – 4; P1 – 3; P2 – 3; P3 – 2
14	P0 – 4; P1 – 7; P2 – 6; P3 – 3	P0 – 4; P1 – 4; P2 – 0; P3 – 7	P0 – 1; P1 – 4; P2 – 3; P3 – 3
15	P0 – 1; P1 – 3; P2 – 2; P3 – 7	P0 – 0; P1 – 4; P2 – 6; P3 – 4	P0 – 2; P1 – 1; P2 – 4; P3 – 1
16	P0 – 5; P1 – 3; P2 – 4; P3 – 2	P0 – 0; P1 – 4; P2 – 8; P3 – 4	P0 – 3; P1 – 1; P2 – 3; P3 – 1
17	P0 – 2; P1 – 4; P2 – 1; P3 – 6	P0 – 3; P1 – 0; P2 – 6; P3 – 7	P0 – 3; P1 – 4; P2 – 1; P3 – 3
18	P0 – 3; P1 – 6; P2 – 3; P3 – 1	P0 – 4; P1 – 1; P2 – 1; P3 – 0	P0 – 1; P1 – 3; P2 – 3; P3 – 1
19	P0 – 10; P1 – 2; P2 – 5; P3 – 3	P0 – 11; P1 – 7; P2 – 3; P3 – 0	P0 – 3; P1 – 3; P2 – 1; P3 – 2
20	P0 – 8; P1 – 3; P2 – 2; P3 –3	P0 – 0; P1 – 4; P2 – 7; P3 – 3	P0 – 2; P1 – 2; P2 – 3; P3 – 1
21	P0 – 3; P1 – 6; P2 – 4; P3 – 2;	P0 – 7; P1 – 0; P2 – 5; P3 – 3	P0 – 4; P1 – 2; P2 – 4; P3 – 1
22	P0 – 7; P1 – 6; P2 – 5; P3 – 3	P0 – 0; P1 – 4; P2 – 0; P3 – 4	P0 – 1; P1 – 3; P2 – 4; P3 – 2
23	P0 – 3; P1 – 7; P2 – 4; P3 – 2	P0 – 6; P1 – 0; P2 – 0; P3 – 7	P0 – 3; P1 – 1; P2 – 2; P3 – 1
24	P0 – 2; P1 – 5; P2 – 1; P3 – 7	P0 – 1; P1 – 8; P2 – 0; P3 –	P0 – 2; P1 – 1; P2 – 2; P3 – 3
25	P0 – 3; P1 – 4; P2 – 7; P3 – 1	P0 – 3; P1 – 4; P2 – 4; P3 – 0	P0 – 1; P1 – 1; P2 – 2; P3 – 2

Для Round Robin (RR) величина кванта времени – 3 для всех вариантов. Для приоритетных алгоритмов меньшее значение соответствует более высокому приоритету.

Лабораторная работа № 6

Программирование планировщиков процессов

Цель работы: изучение алгоритмов программирования планировщиков процессов.

6.1. Контрольное задание

Разработать программу на языке Си++ (или другом), осуществляющую моделирование работы заданного алгоритма планирования. В процессе работы программы на экран должна выводиться следующая информация:

1. Номер текущего кванта времени процессора.
2. Таблица процессов с указанием имени процессов, продолжительности и времени появления процесса, приоритета (в зависимости от задания), оставшегося времени выполнения.
3. Таблица планирования процессов с отображением текущего состояния процессов.

После запуска программа должна в диалоговом режиме запросить следующую информацию о процессах – имя, длительность, приоритет, время появления, для алгоритмов RR – длительность кванта времени. Выполнение должно производиться в **пошаговом режиме** (по нажатию на кнопку). После окончания работы процесса на экране должно выводиться сообщение о его завершении «*Процесс 1 завершил работы*».

Примерный вариант предоставления информации о работе планировщика процессов представлен на рис. 6.1. Некоторые колонки в зависимости от заданного алгоритма могут отсутствовать.

№	Имя	Появление	Длительность	Приоритет	Осталось
1	P6	0	3	3	1
2	P2	2	4	2	3
3	P3	3	5	1	4

Текущий квант времени процесса: 6

№	Имя	0	1	2	3	4	5	6	…	…	…	…	…
1	P6	И	И	И									
2	P2	Г	Г	Г	И	Г	Г	И	И	И	И	И	И
3	P3	Г	Г	Г	Г	И	И						

Рис. 6.1. Пример предоставления информации о работе планировщика

Вариант задания (табл. 6.1) выбирается студентом самостоятельно, оценивая его сложность. Максимальный возможный бал по работе – 10. Дополнительные 2 балла могут быть добавлены преподавателем за:

1) удобный интерфейс с инструментами автоматизации ввода;

2) алгоритмические решения и др.

Таблица 6.1

Варианты заданий

Вариант	**Алгоритм планировщика**	**Сложность, балл**
1	Алгоритм FCFS. Невытесняющий	4
2	Алгоритм FCFS. Невытесняющий, приоритетный	4
3	Алгоритм FCFS. Вытесняющий	5
4	Алгоритм FCFS. Вытесняющий, приоритетный	5
5	Алгоритм SJF. Невытесняющий	5
6	Алгоритм SJF. Невытесняющий, приоритетный	6
7	Алгоритм SJF. Вытесняющий	7
8	Алгоритм SJF. Вытесняющий, приоритетный	7
9	Алгоритм RR	5
10	Алгоритм RR. Невытесняющий, приоритетный	5
11	Алгоритм RR. Невытесняющий, приоритетный (задается длительностью процесса)	6
12	Алгоритм RR. Вытесняющий, приоритетный	7
13	Алгоритм RR. Вытесняющий, приоритетный (задается длительностью процесса)	7
14	Алгоритм RR. Невытесняющий, приоритетный (приоритет увеличивается на единицу при каждом 10 (можно задавать при вводе) кванте состояния «ГОТОВНОСТЬ»))	8
15	Алгоритм RR. Вытесняющий, приоритетный (приоритет увеличивается на единицу при каждом 10 (можно задавать при вводе) кванте состояния «ГОТОВНОСТЬ»))	8

Литература

1. Таненбаум, Э. Архитектура компьютера / Э. Таненбаум, Т. Остин : пер. с англ. Е. Матвеева. – 6-е изд. – СПб. : Питер, 2014. – 811 с.

2. Олссон, Г. Цифровые системы автоматизации и управления / Г. Олссон, Джангуидо Пиани. – СПб. : Нев. Диалект, 2001. – 557 с. : ил.

3. Новиков, Ю. В. Основы микропроцессорной техники : учеб. пособие / Ю. В. Новиков, П. К. Скоробогатов. – 4-е изд., испр. – М. : Интернет-ун-т информ. технологий : Бином. Лаб. знаний, 2011. – 357 с.

4. Степанов, А. Н. Архитектура вычислительных систем и компьютерных сетей : учеб. пособие для вузов / А. Н. Степанов. – СПб. : Питер, 2007. – 508 с.

5. Харазов, В. Г. Интегрированные системы управления технологическими процессами / В. Г. Харазов – СПб. : Профессия, 2009. – 592 с.

6. Сущенко, С. П. Архитектура вычислительных систем / С. П. Сущенко. – Томск : СКК-Пресс, 2006. – 198 с.

7. Максимов, Н. В. Архитектура ЭВМ и вычислительных систем / Н. В. Максимов, Т. Л. Партыка, И. И. Попов. – 5-е изд. – М. : Форум ; Инфра-М, 2013. – 512 с.

8. Мелехин В. Ф. Вычислительные машины, системы и сети / В. Ф. Мелехин, Е. Г. Павловский. – М. : Академия, 2010. – 560 с.

9. Новиков, Ю. В. Основы микропроцессорной техники. Курс лекций : учеб. пособие / Ю. В. Новиков – 2-е изд., испр. – М. : Интернет-ун-т информ. технологий, 2004 – 438 с.

10. Основы микропроцессорной техники. Курс лекций : учеб. пособие / Ю. В. Новиков, П. К. Скоробогатов. – М. : Интернет-ун-т информ. технологий, 2004. – 440 с.

11. Таненбаум, Э. Современные операционные системы / Э. Таненбаум. – 3-е изд. – СПб. : Питер, 2010. – 1115 с.

12. Хартов, В. Я. Микропроцессорные системы : учеб. пособие для вузов / В. Я. Хартов. – М. : Академия, 2010. – 351 с

13. Руководство пользователя TRACE MODE 6 БЫСТРЫЙ СТАРТ / AdAstra Research Group, Ltd. – М. : TRACE MODE, 2012. – 168 с.

14. Цилькер, Б. Я. Организация ЭВМ и систем : учеб. для вузов / Б. Я. Цилькер, С. А. Орлов. – СПб. : Питер, 2004. – 668 с.

15. Горнец, Н. Н. Организация ЭВМ и систем : учеб. пособие для вузов / Н. Н. Горнец. – 2-е изд., стер. – М. : Академия, 2008. – 316 с.

Содержание

Учебное электронное издание комбинированного распространения

Учебное издание

Литвинов Дмитрий Александрович

АППАРАТНО-ПРОГРАММНЫЕ СРЕДСТВА ВЫЧИСЛИТЕЛЬНЫХ МАШИН И СИСТЕМ

Практикум по выполнению лабораторных работ по дисциплине «Вычислительные машины и системы» для студентов специальности 1-53 07 01 «Промышленная электроника» дневной формы обучения

Электронный аналог печатного издания

Редактор *Т. Н. Мисюрова*
Компьютерная верстка *И. П. Минина*

Подписано в печать 02.04.19.
Формат 60x84/$_{16}$. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 2,79. Уч.-изд. л. 3,05.
Изд. № 28.
http://www.gstu.by

Издатель и полиграфическое исполнение
Гомельский государственный
технический университет имени П. О. Сухого.
Свидетельство о гос. регистрации в качестве издателя
печатных изданий за № 1/273 от 04.04.2014 г.
пр. Октября, 48, 246746, г. Гомель