МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)»
ВОЛЖСКИЙ ФИЛИАЛ

Кафедра гуманитарных и естественнонаучных дисциплин

Методические указания к лабораторным работам
по дисциплине

ЭВМ И ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА

Направление подготовки

09.03.01 Информатика и вычислительная техника

Направленность (профиль, специализация) образовательной программы

«Автоматизированные системы обработки информации и управления»

Квалификация

бакалавр

Чебоксары
2019

СОДЕРЖАНИЕ

ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Все лабораторные работы курса “Интернет программирование” выполняются в едином порядке, в соответствии с едиными требованиями.

Порядок выполнения работы.

1. Изучить "Краткие теоретические сведения".
2. Получить у преподавателя условия задач (номер варианта).
3. Решить задачи (см. п.
4. **Порядок решения задачи.**).
5. Показать работу программ преподавателю.
6. Распечатать тексты полученных программ.
7. Оформить отчет (пояснительную записку) (см. п.
8. **Требования к отчету**).
9. Сдать отчет (пояснительную записку) преподавателю.
10. Подготовиться к ответам на контрольные вопросы.
11. Защитить работу.

Порядок решения задачи.

1. Изучить условие задачи.
2. Разработать алгоритм решения задачи.
3. Согласовать алгоритм решения задачи с ведущим преподавателем
4. Разработать порядок работы с программой.
5. Написать и ввести программу.
6. Отладить программу.
7. Скорректировать алгоритм и порядок работы программы по результатам отладки.
8. Ответить на вопросы задания (если есть).

Общие требования к программам.

- Программа должна выводить на терминал реквизиты авторов (фамилию, имя и группу).
- Программа, использующая ввод с клавиатуры, должна подсказывать пользователю, что ему делать.

Требования к отчету.

Требования к оформлению.

1. Отчет выполняется на листах формата А4 с использованием любого текстового процессора и распечатывается на принтере.
2. Титульный лист отчета выполняется по стандартной форме (см. Рисунок 1).
3. Рамка на последующих листах отчета необязательна.

4. Листы отчета необходимо скрепить.
5. Отчет должен быть подписан исполнителем.
6. Тексты программ должны содержать комментарии к использованию переменных и работе программы.
7. Тексты программ распечатываются на принтере.

Содержание отчета.

1. Титульный лист (см. Рисунок 1).
2. Цель работы.
3. Описание решений задач (см. п. 0).
4. Тексты программ (распечатки).

Порядок описания решения задачи.

Решение каждой задачи описывается в следующем порядке;

1. Условие задачи.
2. Физическое и математическое решение задачи.

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)»
Волжский филиал

Факультет очный
Кафедра математики и информатики
Направление подготовки Автоматизированные системы обработки информации и управления

О Т Ч Е Т

к лабораторной работе №1

«Доступ к базам данных. СУБД MySQL»

Дата сдачи «__»_____20__г
Зачтено «__»_____20__г

Выполнил: студент гр. ОП-14,
Петров И.В.
Преподаватель: Семенов Б.И.

Чебоксары 2015

Рисунок 1. Пример оформления титульного листа.

ТЕМАТИКА ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа №1. Принципы и структуры вычислительных машин и периферийных устройств.

Цель работы

Освоить создание программ простой структуры в функциональном языке высокого уровня. Подготовить общие черты программ простой структуры на языке программирования Assembler.

Основные сведения

1. Основы работы с Visual Studio C++

Основные способы создания программы на языке высокого уровня C++.

Общие сведения:

Вычислительная машина — это комплекс технических и программных средств, предназначенный для автоматизации подготовки и решения задач пользователей.

Вычислительная система — это совокупность взаимосвязанных и взаимодействующих процессоров или вычислительных машин, периферийного оборудования и программного обеспечения, предназначенную для подготовки и решения задач пользователей.

Под архитектурой вычислительной машины обычно понимается логическое построение ВМ, то есть то, какой машина представляется программисту. Из рассмотрения выпадают вопросы физического построения вычислительных средств: состав устройств, число регистров процессора, емкость памяти, наличие специального блока для обработки вещественных чисел, тактовая частота центрального процессора и т.д. Этот круг вопросов принято определять понятием организация вычислительной машины.

Уровни детализации структуры вычислительной машины

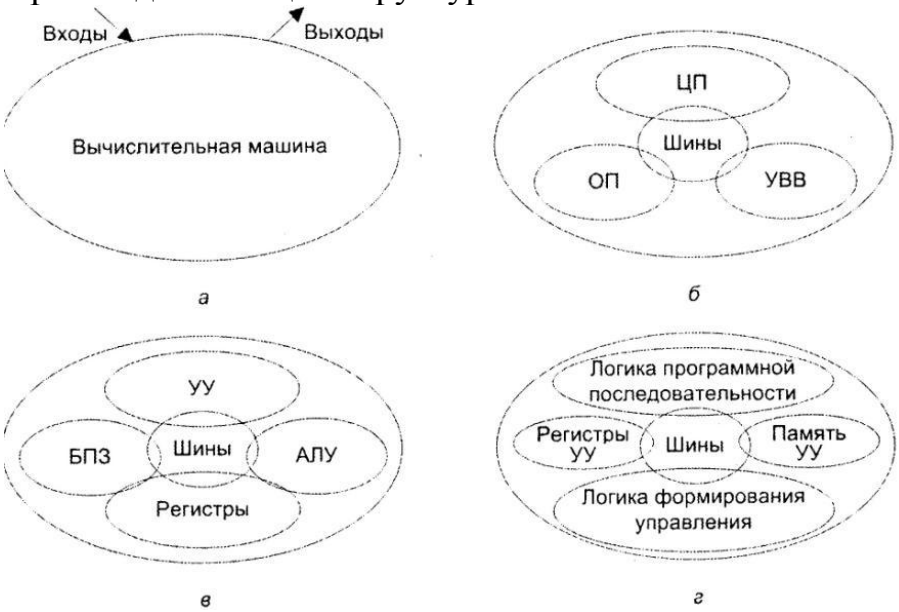

Рис. 1.1. Уровни детализации вычислительной машины:

а — уровень «черного ящика»;

б — уровень общей архитектуры;

в — уровень архитектуры центрального процессора;

г — уровень архитектуры устройства управления

На первом уровне вычислительная машина рассматривается как устройство, способное хранить и обрабатывать информацию, а также обмениваться данными с внешним миром (см. рис. 1.1, а). ВМ представляется «черным ящиком», который может быть подключен к коммуникационной сети и к которому, в свою очередь, могут подсоединяться периферийные устройства.

Уровень общей архитектуры (см. рис. 1.1,6) предполагает представление ВМ в виде четырех составляющих: центрального процессора (ЦП), основной памяти (ОП), устройства ввода/вывода (УВВ) и системы шин.

На третьем уровне детализируется каждое из устройств второго уровня. Для примера взят центральный процессор (см. рис. 1.1, в). В простейшем варианте в нем можно выделить: арифметико-логическое устройство (АЛУ), обеспечивающее обработку целых чисел; блок обработки чисел в формате с плавающей запятой (БПЗ); регистры процессора, использующиеся для краткосрочного хранения команд, данных и адресов; устройство управления (УУ), обеспечивающее совместное функционирование устройств ВМ; внутренние шины.

На четвертом уровне детализируются элементы третьего уровня. Так, на рис. 1.1, г раскрыта структура устройства управления. УУ представлено в виде четырех составляющих: логики программной последовательности — электронных схем, обеспечивающих выполнение команд программы в последовательности, предписываемой программой; регистров и дешифраторов устройства управления; управляющей памяти; логики формирования управления, генерирующей все необходимые управляющие сигналы.

Применительно к параллельным и распределенным многопроцессорным и многомашинным вычислительным системам зачастую вводят понятие «метауровня»

Список индивидуальных заданий

1. Напишите программу «Hello, world!» и преобразуйте ее в программу «Привет, мир!».
2. Напишите программу для перевода температуры в градусах по Фаренгейту в градусы по Цель- сию по формуле C = 5/9 (F – 32).
3. Напишите программу для вычисления площади треугольника по трем сторонам.
4. Заданы моменты начала и конца некоторого промежутка времени в часах, минутах и секундах (в пределах одних суток). Найти продолжительность этого промежутка в тех же единицах..

Лабораторная работа №2. Архитектура системы команд.

Цель работы: выработать практические навыки работы с изучить встроенные и внешние объекты. научиться создавать, вводить в компьютер, выполнять и исправлять простейшие программы на языке в режиме диалога, познакомиться с диагностическими сообщениями компилятора об ошибках при выполнении программ, реализующих линейные алгоритмы

Общие сведения:

SIMD-расширения (Single Instruction Multiple Data) были введены в архитектуру x86 с целью повышения скорости обработки потоковых данных. Основная идея заключается в одновременной обработке нескольких элементов данных за одну инструкцию.

1.1. Расширение MMX

Первой SIMD-расширение в свой x86-процессор ввела фирма Intel – это расширение MMX. Оно стало использоваться в процессорах Pentium MMX (расширение архитектуры Pentium или P5) и Pentium II (расширение архитектуры Pentium Pro или P6). Расширение MMX работает с 64-битными регистрами MM0-MM7, физически расположенными на регистрах сопроцессора, и включает 57 новых инструкций для работы с ними. 64-битные регистры логически могут представляться как одно 64-битное, два 32-битных, четыре 16-битных или восемь 8-битных упакованных целых. Еще одна особенность технологии MMX – это арифметика с насыщением. При этом переполнение не является циклическим, как обычно, а фиксируется минимальное или максимальное значение. Например, для 8-битного беззнакового целого x:

обычная арифметика: x=254; x+=3; // результат x=1
арифметика с насыщением: x=254; x+=3; // результат x=255

1.2. Расширение 3DNow!

Технология 3DNow! была введена фирмой AMD в процессорах K6-2. Это была первая технология, выполняющая потоковую обработку вещественных данных. Расширение работает с регистрами 64-битными MMX, которые представляются как два 32-битных вещественных числа с одинарной точностью. Система команд расширена 21 новой инструкцией, среди которых есть команда выборки данных в кэш L1. В процессорах Athlon и Duron набор инструкций 3DNow! был несколько дополнен новыми инструкциями для работы с вещественными числами, а также инструкциями MMX и управления кэшированием.

1.3. Расширение SSE

С процессором Intel Pentium III впервые появилось расширение SSE (Streaming SIMD Extension). Это расширение работает с независимым блоком из восьми 128-битных регистров XMM0-XMM7. Каждый регистр XMM представляет собой четыре упакованных 32-битных вещественных числа с одинарной точностью. Команды блока XMM позволяют выполнять как векторные (над всеми четырьмя значениями регистра), так и скалярные операции (только над одним самым младшим значением). Кроме инструкций с блоком XMM в расширение SSE входят и дополнительные целочисленные инструкции с регистрами MMX, а также инструкции управления кэшированием.

1.4. Расширение SSE2

В процессоре Intel Pentium 4 набор инструкций получил очередное расширение – SSE2. Оно позволяет работать с 128-битными регистрами XMM как с парой упакованных 64-битных вещественных чисел двойной точности, а также с упакованными целыми числами:

16 байт, 8 слов, 4 двойных слова или 2 учетверенных (64-битных) слова. Введены новые инструкции вещественной арифметики двойной точности, инструкции целочисленной арифметики, 128-разрядные для регистров XMM и 64-разрядные для регистров MMX. Ряд старых инструкций MMX распространили и на XMM (в 128-битном варианте). Кроме того, расширена поддержка управления кэшированием и порядком исполнения операций с памятью.

1.5. Расширение SSE3

Дальнейшее расширение системы команд – SSE3 – вводится в процессоре Intel Pentium 4 с ядром Prescott. Это набор из 13 новых инструкций, работающих с блоками XMM, FPU, в том числе двух инструкций, повышающих эффективность синхронизации потоков, в частности, при использовании технологии Hyper-Threading.

1.6. Поддержка SIMD-расширений архитектурой x86-64

Процессоры AMD Athlon64 и AMD Opteron с архитектурой x86-64 поддерживают все выше перечисленные SIMD-расширения, кроме SSE3. Кроме того, число XMM регистров у этих процессоров увеличилось до 16 (XMM0-XMM15). Подробное описание типов и команд SSE приведено в приложении.

2. Встроенные функции потокового SIMD расширения

Типы данных

Для работы с векторными данными, содержащими несколько упакованных значений, используются следующие типы:

__m64 – 64-бит (регистр MMX)
- 1 * 64-битное целое
- 2 * 32-битных целых
- 4 * 16-битных целых
- 8 * 8-битных целых.

__m128 – 128-бит (регистр XMM):
- 4 * 32-битных вещественных (SSE),
- 2 * 64-битных вещественных (SSE2),
- 2 * 64-битное целых (SSE2),
- 4 * 32-битных целых (SSE2),
- 8 * 16-битных целых (SSE2),
- 16 * 8-битных целых (SSE2).

Для наибольшей эффективности элементы таких типов данных должны быть выровнены в памяти по соответствующей границе. Например, начало массива элементов типа __m64 выравнивается по 8 байтам, а массив элементов __m128 – по 16 байтам. Статические переменные и массивы компилятор выравнивает автоматически. Динамические данные компилятор обычно выравнивает по только величине 4 байта. Если данные векторных типов оказались невыровненными, то для работы с ними следует применять специальные команды невыровненного чтения и записи (они работают медленнее обычных – выровненных). Для выделения памяти с выравниванием используется функция:

```
void *_mm_malloc(int size, int align)
```

size – объем выделяемой памяти в байтах (как в malloc),
align – выравнивание в байтах.

Для освобождения памяти, выделенной таким образом, используется функция:

```
void _mm_free(void *p);
```

Например:

```
float *x;       // массив для обработки с помощью инструкций SSE
x=(float)_mm_malloc(N*sizeof(float),16);
// … здесь обработка …
_mm_free(x);
```

Встроенные функции SSE для работы с вещественными числами

Заголовочный файл xmmintrin.h содержит объявления встроенных функций (intrinnsics) SSE.

Арифметические функции

Функция	**Инструкция**	**Операция**	**R0**	**R1**	**R2**	**R3**
_mm_add_ss	ADDSS	сложение	a0 [op] b0	a1	a2	a3
_mm_add_ps	ADDPS	сложение	a0 [op] b0	a1 [op] b1	a2 [op] b2	a3 [op] b3
_mm_sub_ss	SUBSS	вычитание	a0 [op] b0	a1	a2	a3
_mm_sub_ps	SUBPS	вычитание	a0 [op] b0	a1 [op] b1	a2 [op] b2	a3 [op] b3
_mm_mul_ss	MULSS	умножение	a0 [op] b0	a1	a2	a3
_mm_mul_ps	MULPS	умножение	a0 [op] b0	a1 [op] b1	a2 [op] b2	a3 [op] b3
_mm_div_ss	DIVSS	деление	a0 [op] b0	a1	a2	a3
_mm_div_ps	DIVPS	деление	a0 [op] b0	a1 [op] b1	a2 [op] b2	a3 [op] b3
_mm_sqrt_ss	SQRTSS	квадратный корень	[op] a0	a1	a2	a3
_mm_sqrt_ps	SQRTPS	квадратный корень	[op] a0	[op] b1	[op] b2	[op] b3
_mm_rcp_ss	RCPSS	обратное значение	[op] a0	a1	a2	a3
_mm_rcp_ps	RCPPS	обратное значение	[op] a0	[op] b1	[op] b2	[op] b3
_mm_rsqrt_ss	RSQRTSS	обратное значение квадратного корня	[op] a0	a1	a2	a3
_mm_rsqrt_ps	RSQRTPS	обратное значение квадратного корня	[op] a0	[op] b1	[op] b2	[op] b3
_mm_min_ss	MINSS	минимум	[op](a0,b0)	a1	a2	a3
_mm_min_ps	MINPS	минимум	[op](a0,b0)	[op](a1,b1)	[op](a2,b2)	[op](a3,b3)
_mm_max_ss	MAXSS	максимум	[op](a0,b0)	a1	a2	a3
_mm_max_ps	MAXPS	максимум	[op](a0,b0)	[op](a1,b1)	[op](a2,b2)	[op](a3,b3)

Функции сравнения

Каждая встроенная функция сравнения выполняет сравнение операндов a и b. В векторной форме сравниваются четыре вещественных значения параметра a с четырьмя вещественными значениями параметра b, и возвращается 128-битная маска. В скалярной форме сравниваются младшие значения параметров, возвращается 32-битная маска, остальные три старших значения копируются из параметра a. Маска устанавливается в значение `0xffffffff для тех элементов, результат сравнения которых истина, и 0x0, где результат сравнения ложь.`

Имя	Сравнение	Инструкция
_mm_cmpeq_ss	равно	CMPEQSS
_mm_cmpeq_ps	равно	CMPEQPS
_mm_cmplt_ss	меньше	CMPLTSS
_mm_cmplt_ps	меньше	CMPLTPS
_mm_cmple_ss	меньше или равно	CMPLESS
_mm_cmple_ps	меньше или равно	CMPLEPS
_mm_cmpgt_ss	больше	CMPLTSS
_mm_cmpgt_ps	больше	CMPLTPS
_mm_cmpge_ss	больше или равно	CMPLESS
_mm_cmpge_ps	больше или равно	CMPLEPS

_mm_cmpneq_ss	не равно	CMPNEQSS
_mm_cmpneq_ps	не равно	CMPNEQPS
_mm_cmpnlt_ss	не меньше	CMPNLTSS
_mm_cmpnlt_ps	не меньше	CMPNLTPS
_mm_cmpnle_ss	не меньше или равно	CMPNLESS
_mm_cmpnle_ps	не меньше или равно	CMPNLEPS
_mm_cmpngt_ss	не больше	CMPNLTSS
_mm_cmpngt_ps	не больше	CMPNLTPS
_mm_cmpnge_ss	не больше или равно	CMPNLESS
_mm_cmpnge_ps	не больше или равно	CMPNLEPS
_mm_cmpord_ss	упорядочены	CMPORDSS
_mm_cmpord_ps	упорядочены	CMPORDPS
_mm_cmpunord_ss	неупорядочены	CMPUNORDSS
_mm_cmpunord_ps	неупорядочены	CMPUNORDPS
_mm_comieq_ss	равно	COMISS
_mm_comilt_ss	меньше	COMISS
_mm_comile_ss	меньше или равно	COMISS
_mm_comigt_ss	больше	COMISS
_mm_comige_ss	большеили равно	COMISS
_mm_comineq_ss	не равно	COMISS
_mm_ucomieq_ss	равно	UCOMISS
_mm_ucomilt_ss	меньше	UCOMISS
_mm_ucomile_ss	меньше или равно	UCOMISS
_mm_ucomigt_ss	больше	UCOMISS
_mm_ucomige_ss	больше или равно	UCOMISS
_mm_ucomineq_ss	не равно	UCOMISS

Операции преобразования типов

Имя функции	**Операция**	**Инструкция**
_mm_cvtss_si32	Преобразует младший float в 32-битное целое	CVTSS2SI
_mm_cvtps_pi32	Преобразует два младших float в два упакованных 32-битных целых	CVTPS2PI
_mm_cvttss_si32	Преобразует младший float в 32-битное целое, отбрасывая дробную часть	CVTTSS2SI
_mm_cvttps_pi32	Преобразует два младших float в два упакованных 32-битных целых, отбрасывая дробную часть	CVTTPS2PI
_mm_cvtsi32_ss	Преобразует 32-битное целое в float	CVTSI2SS
_mm_cvtpi32_ps	Преобразует два упакованных 32-битных целых в два младших float	CVTTPS2PI
_mm_cvtpi16_ps	Преобразует четыре упакованных 16-битных целых в упакованные float	составная
_mm_cvtpu16_ps	Преобразует четыре упакованных беззнаковых 16-битных целых в упакованные float	составная
_mm_cvtpi8_ps	Преобразует четыре младших упакованных 8-битных целых в четыре упакованных float	составная

_mm_cvtpu8_ps	Преобразует четыре младших упакованных беззнаковых 8-битных целых в четыре упакованных float	составная
_mm_cvtpi32x2_ps	Преобразует две пары упакованных 32-битных целых в четыре упакованных float	составная
_mm_cvtps_pi16	Преобразует четыре упакованных float в четыре 16-битных целых	составная
_mm_cvtps_pi8	Преобразует четыре упакованных float в четыре младших 8-битных целых	составная

Другие функции

Имя функции	**Операция**	**Инструкция**
_mm_shuffle_ps	перестановка упакованных значений	SHUFPS
_mm_shuffle_pi16	перестановка упакованных значений	PSHUFW
_mm_unpackhi_ps	выборка старших значений	UNPCKHPS
_mm_unpacklo_ps	выборка младших значений	UNPCKLPS
_mm_loadh_pi	загрузка старших значений	MOVHPS reg, mem
_mm_storeh_pi	сохранение старших значений	MOVHPS mem, reg
_mm_movehl_ps	копирование старшей половины в младшую	MOVHLPS
_mm_movelh_ps	копирование младшей половины в старшую	MOVLHPS
_mm_loadl_pi	загрузка младших значений	MOVLPS reg, mem
_mm_storel_pi	сохранение младших значений	MOVLPS mem, reg
_mm_movemask_ps	создание знаковой маски	MOVMSKPS
_mm_getcsr	сохранить регистр состояния	STMXCSR
_mm_setcsr	установить регистр состояния	LDMXCSR

Команды для инициализации и работы с памятью

Инициализация памяти

Имя функции	Операция	Инструкция
_mm_load_ss	загрузить младшее значение и очистить остальные три значения	MOVSS
_mm_load1_ps	загрузить одно значение во все четыре позиции	MOVSS + Shuffling
_mm_load_ps	Загрузить четыре значения по выровненному адресу	MOVAPS
_mm_loadu_ps	Загрузить четыре значения по невыровненному адресу	MOVUPS
_mm_loadr_ps	Загрузить четыре значения в обратном порядке	MOVAPS + Shuffling

Инициализация значений

Имя функции	**Операция**	**Инструкция**
_mm_set_ss	устанавливает самое младшее значение и обнуляет три остальных	составная
_mm_set1_ps	устанавливает четыре позиции в одно значение	составная
_mm_set_ps	устанавливает четыре значения, выровненные по адресу	составная

_mm_setr_ps	устанавливает четыре значения в обратном порядке	составная
_mm_setzero_ps	Обнуляет все четыре значения	составная

Операции записи

Имя функции	Операция	Инструкция
_mm_store_ss	записать младшее значение	MOVSS
_mm_store1_ps	записать младшее значение во все четыре позиции	MOVSS + Shuffling
_mm_store_ps	записать четыре значения по выровненному адресу	MOVAPS
_mm_storeu_ps	записать четыре значения по невыровненному адресу	MOVUPS
_mm_storer_ps	записать четыре значения в обратном порядке	MOVAPS + Shuffling
_mm_move_ss	записать младшее значение и оставить без изменения три остальных значения	MOVSS

Поддержка кэш-памяти в SSE

Имя функции	Операция	Инструкция
_mm_prefetch	Загружает одну кэш-строку по указанному адресу в кэш-память	PREFETCH
_mm_stream_pi	Записывает данные в память без записи в кэш	MOVNTQ
_mm_stream_ps	Записывает данные в память без записи в кэш по адресу, выровненному по 16 байт	MOVNTPS
_mm_sfence	Гарантирует, что все предшествующие записи в память завершатся до следующей записи.	SFENCE

3. Использование встроенных функций SSE в программе на языке Си

```
// скалярное произведение векторов
#include <stdio.h>
#include <xmmintrin.h>
#define N 10000000

// «обычная» функция
float inner1(float *x,float *y,int n)
{
	float s;
	int i;
	s=0;
	for(i=0;i<n;i++)
		s+=x[i]*y[i];
	return s;
}

// функция с использованием SSE intrinsics
float inner2(float *x,float *y,int n)
{
	float sum;
	int i;
	__m128 *xx,*yy;
	__m128 p,s;
	xx=(__m128 *)x;
	yy=(__m128 *)y;
	s=_mm_set_ps1(0);
	for (i=0;i<n/4;i++)
	  {
	    p=_mm_mul_ps(xx[i], yy[i]); // векторное умножение четырех чисел
	    s=_mm_add_ps(s,p);          // векторное сложение четырех чисел
	  }
	p=_mm_movehl_ps(p,s); // перемещение двух старших значений s в младшие p
	s=_mm_add_ps(s,p);    // векторное сложение
```

```
	p=_mm_shuffle_ps(s,s,1);//перемещение второго значения в s в младшую позицию в p
	s=_mm_add_ss(s,p);    // скалярное сложение
	_mm_store_ss(&sum,s); // запись младшего значения в память
	return sum;
}

int main()
{
	float *x,*y, s;
	long t;
	int i;
	// выделение памяти с выравниванием
	x=(float *)_mm_malloc(N*sizeof(float),16);
	y=(float *)_mm_malloc(N*sizeof(float),16);
	for (i=0;i<N;i++)
	{
		x[i]=10*i/N;
		y[i]=10*(N-i-1)/N;
	}

	// Using x87
	s=inner1(x,y,N);
	printf("Result: %f\n",s);

	// Using SSE
	s=inner2(x,y,N);
	printf("Result: %f\n",s);
	_mm_free(x);
	_mm_free(y);
	return 0;
}
```

Задание.

1. Реализовать процедуру умножения квадратных матриц (размером кратным четырём) без использования специальных расширений и с использованием расширений SSE, сравнить время выполнения этих реализаций (Обязательное задание – 10 баллов).

2. В соответствии с вариантом задания реализовать матрично-векторную (с одинаковым размером матриц и векторов кратным четырём) процедуру с использованием расширений SSE (Дополнительное задание – 7 баллов).

3. С использованием инструкции *cpuid* определить наличие расширения SSE (Дополнительное задание – 3 балла).

4. Крайний срок сдачи – **7 мая 2011 года**.

Варианты.

В предложенных вариантах предполагается, что α, β – скаляры, x, y – векторы, A, B – матрицы:

1. Операция $aAx + \beta By$.

2. Операция $Ax + \beta By$.

3. Операция $aAx + By$.

4. Операция $aAx + y$.

5. Операция $ax + \beta By$.

6. Операция $aAx - \beta By$.

7. Операция $Ax - \beta By$.

8. Операция $aAx - By$.

9. Операция $aAx - y$.

10. Операция $ax - \beta By$.

Контрольные вопросы:

Решить указанные в варианте задачи, используя основные операторы языка С++. При решении задачи, использовать все типы циклов (for, while, do while).

2. Дана последовательность из n целых чисел. Найти среднее арифметическое этой последовательности.

3. Дана последовательность из n целых чисел. Найти сумму четных элементов этой последовательности.

4. Дана последовательность из n целых чисел. Найти сумму элементов с четными номерами из этой последовательности.

5. Дана последовательность из n целых чисел. Найти сумму нечетных элементов этой последовательности.

6. Дана последовательность из n целых чисел. Найти сумму элементов с нечетными номерами из этой последовательности.

7. Дана последовательность из n целых чисел. Найти минимальный элемент в этой последовательности.

8. Дана последовательность из n целых чисел. Найти номер максимального элемента в этой последовательности.

9. Дана последовательность из n целых чисел. Найти номер минимального элемента в этой последовательности.

10. Дана последовательность из n целых чисел. Найти максимальный элемент в этой последовательности.

11. Дана последовательность из n целых чисел. Найти сумму минимального и максимального элементов в этой последовательности.

12. Дана последовательность из n целых чисел. Найти разность минимального и максимального элементов в этой последовательности.

13. Дана последовательность из n целых чисел. Найти количество нечетных элементов этой последовательности.

14. Дана последовательность из n целых чисел. Найти количество четных элементов этой последовательности.

15. Дана последовательность из n целых чисел. Найти количество элементов этой последовательности, кратных числу К.

16. Дана последовательность из n целых чисел. Найти количество элементов этой последовательности, кратных ее первому элементу.

17. Дана последовательность из n целых чисел. Найти количество элементов этой последовательности, кратных числу K1 и не кратных числу K2.

18. Дана последовательность из n целых чисел. Определить, каких чисел в этой последовательности больше: положительных или отрицательных.

19. Дана последовательность целых чисел, за которой следует 0. Найти среднее арифметическое этой последовательности.

20. Дана последовательность целых чисел, за которой следует 0. Найти сумму четных элементов этой последовательности.

21. Дана последовательность целых чисел, за которой следует 0. Найти сумму элементов с четными номерами из этой последовательности.

22. Дана последовательность целых чисел, за которой следует 0. Найти сумму нечетных элементов этой последовательности.

23. Дана последовательность целых чисел, за которой следует 0. Найти сумму элементов с нечетными номерами из этой последовательности.

24. Дана последовательность целых чисел, за которой следует 0. Найти минимальный элемент в этой последовательности.

25. Дана последовательность целых чисел, за которой следует 0. Найти номер максимального элемента в этой последовательности.

26. Дана последовательность целых чисел, за которой следует 0. Найти номер минимального элемента в этой последовательности.

27. Дана последовательность целых чисел, за которой следует 0. Найти максимальный элемент в этой последовательности.

28. Дана последовательность целых чисел, за которой следует 0. Найти сумму минимального и максимального элементов в этой последовательности.

29. Дана последовательность целых чисел, за которой следует 0. Найти разность минимального и максимального элементов в этой последовательности.

30. Дана последовательность целых чисел, за которой следует 0. Найти количество нечетных элементов этой последовательности.

31. Дана последовательность целых чисел, за которой следует 0. Найти количество четных элементов этой последовательности.

32. Дана последовательность целых чисел, за которой следует 0. Найти количество элементов этой последовательности, кратных числу K.

33. Дана последовательность целых чисел, за которой следует 0. Найти количество элементов этой последовательности, кратных ее первому элементу.

34. Дана последовательность целых чисел, за которой следует 0. Найти количество элементов этой последовательности, кратных числу K1 и не кратных числу K2.

35. Дана последовательность целых чисел, за которой следует 0. Определить, каких чисел в этой последовательности больше: положительных или отрицательных. , всего n слагаемых; S=1+3+5+7+ …, всего n слагаемых; S=1+2-3+4+5-6+7+8-9+…, всего n слагаемых; S=15+17-19+21+23-25+…, всего n слагаемых;

Лабораторная работа №3. Типы и форматы числовых операндов.

Цель работы: научиться правильно использовать условный оператор if; научиться составлять программы решения задач на разветвляющиеся алгоритмы.

Общие сведения.

1. Представление беззнаковых целых чисел

Для представления беззнаковых целых чисел необходимо перевести из десятичной системы исчисления в двоичную. Например, число 123_{10} можно представить в виде суперпозиции по степеням двойки:

$$123_{10} = 64_{10} + 32_{10} + 16_{10} + 8_{10} + 2_{10} + 1_{10} = 2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 = 1111011_2,$$

где единицы в двоичном представлении числа стоят на позиции соответствующих степеней двоек. Для простоты рассмотрим беззнаковый однобайтовый тип данных (тип *unsigned char* в языках C/C++). Для записи числа 123 в такой тип данных необходимо дополнить двоичное представление до 8 знаков $123_{10} = 01111011_2$ и записать полученные значения в соответствующие биты:

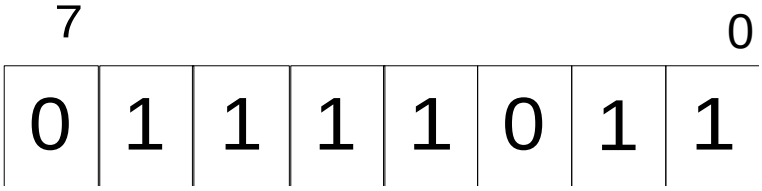

Рисунок 1.

2. Представление знаковых целых чисел

В случае знаковых типов данных старший бит отвечает за знак числа (1 – отрицательное число, 0 – положительное число). Основной проблемой является представление отрицательных чисел. Для такого представления существует следующий алгоритм:

1. нахождение двоичного представления модуля числа,
2. нахождение двоичного дополнения числа,
3. прибавление единицы.

Рассмотрим алгоритм на примере. Представим число -84 в знаковом однобайтном типе данных (тип *char* в языках C/C++):

1. Двоичное представление $|-84_{10}| = 84_{10} = 2^6 + 2^4 + 2^2 = 01010100_2$,
2. Для нахождения двоичного представления инвертируем все биты числа $01010100_2 \to 10101011_2$,
3. Прибавляем единицу $10101011_2 + 1_2 = 10101100_2$.

После этого записываем полученные значения в соответствующие биты:

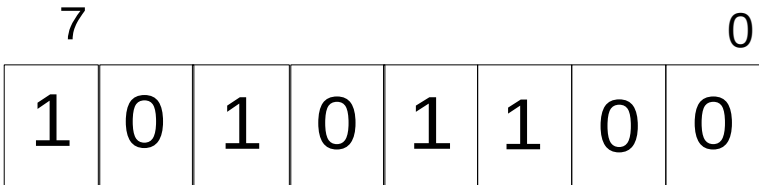

Рисунок 2.

3. Представление вещественных чисел

Основной интерес в вычислениях представляют вещественные типы данных и погрешности округления, связанные с ними. По стандарту IEEE 754 вещественное число A представляется в виде:

$$A = (-1)^S \times M \times 2^E$$

$$1 \le M < 2$$

Рисунок 3.

Где S – однобитовый знак числа, M – нормализованная мантисса, E – показатель степени двойки. В случае типа *float* под мантиссу выделяется 23 бита, экспоненту 8 бит, в случае типа *double* 52 бита, экспоненту 11 бит.

Приведём пример представления вещественного числа 0.15625 в типе *float*. Основной задачей является запись числа в виде $A = (-1)^S \times M \times 2^E$. Число можно записать в виде $0.15625 = 1.25 \times 2^{-3}$, в данном случае мантисса имеет вид $M = 1.25_{10} = 1.01_2$ нормализация мантиссы позволяет отбросить единицу и записывать только дробную часть. Таким образом $fraction = 01_2$. Далее записываем показатель степени двойки. При этом нужно учитывать, что эта степень может быть как отрицательной так и положительной. Для этого показатель степени имеет вид:

$$exponent = E + \left(2^{q-1} - 1\right)_{10} = -3_{10} + 127_{10} = 124_{10} = 01111100_2,$$

где q – количество бит на показатель степени двойки. В результате число 0.15625 представимо в виде:

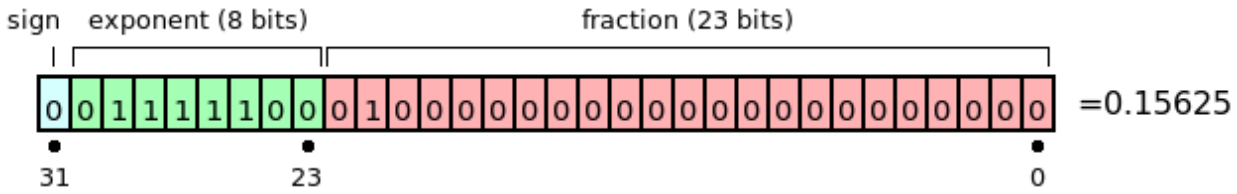

Рисунок 4.

В завершении описания представления вещественных чисел нужно отметить, что для стандартных типов данных (*float* и *double*) имеют место следующие значения:

Таблица 1.

Тип	Минимальный порядок*	Максимальный порядок	Число значащих знаков
float	– 45	38	7
double	– 323	308	15

(*) Стоит отметить, что мантисса может быть ненормализованной, что и приводит к таким значениям минимального порядка.

4. Идентификация оборудования и программного окружения

Средствами операционной системы Windows можно узнать достаточно много информации об оборудовании, памяти (функция *GlobalMemoryStatus*), жёстких дисках (функция *GetDiskFreeSpace*), сети и мониторе (функция *GetSystemMetrics* при различных параметрах), программном окружении (функции *GetComputerName* и *GetUserName*) и о многом другом. Подробное описание функций и примеры их использования можно найти в справочной системе MSDN.

Для определения таких параметров процессора, как фирма производитель, наличие расширений, количества и параметров кэшей команд и данных, TLB и других параметров в случае архитектур x86 используется инструкция процессора *cpuid*, которая имеет интерфейс на языке C/C++ *__cpuid*. Так для определения идентификатора процессора имеет место следующий код:

```
#include <intrin.h>              // подключение описания функции __cpuid
…
int CPUInfo[4];
char CPUString[32];
__cpuid(CPUInfo, 0);
memset(CPUString, 0, sizeof(CPUString));
*((int*)CPUString)     = CPUInfo[1];
*((int*)(CPUString+4)) = CPUInfo[3];
*((int*)(CPUString+8)) = CPUInfo[2];
printf("  CPU vendor: %s\n",CPUString);
…
```

Первый параметр функции *__cpuid* – 4-х элементный целочисленный массив, который соответствует регистрам eax, ebx, ecx, edx после выполнения инструкции. Второй параметр функции – номер функции инструкции. Подробная информация о номерах функций инструкции *cpuid* и содержимом регистров приведена в документах [1,2] для процессоров Intel и AMD. Так например с помощью функций 0x80000002, 0x80000003, 0x80000004 можно узнать полное название процессора.

Задание.

1. В соответствии с вариантом задания записать представление целого числа в типе *char* и вещественного числа в типе *float* (Обязательное задание – 5 баллов).

2. С помощью функций WinAPI определить информацию об оперативной памяти (Дополнительное задание – 1 балл).

3. С помощью функций WinAPI определить информацию о памяти на одном из жёстких дисков (Дополнительное задание – 2 балла).

4. С помощью инструкции cpuid определить название процессора (Дополнительное задание – 2 балла).

Варианты.

1. Целое число –12, вещественное число 12.5.

2. Целое число –23, вещественное число 12.125.
3. Целое число –56, вещественное число 12.25.
4. Целое число –78, вещественное число 12.75.
5. Целое число –89, вещественное число 12.625.
6. Целое число –90, вещественное число 24.5.
7. Целое число –21, вещественное число 24.125.
8. Целое число –45, вещественное число 24.25.
9. Целое число –78, вещественное число 24.75.
10. Целое число –86, вещественное число 24.625.

Лабораторная работа №4. Типы команд.

Цель работы:

изучение средств языка C++, используемых при наследовании классов.

Общие сведения

Директивы символических определений могут быть использованы для того, чтобы резервировать пространство памяти, поставить в соответствие символическим именам определённые числовые значения, регистры процессора и сегменты. Эти директивы требуют, чтобы имя символа было определено наряду с адресом, числовым значением, регистром или типом сегмента.

Директива Описание

BIT Определяет символическое имя, ссылающееся на адрес бита.

CODE Определяет символическое имя, ссылающееся на адрес кода.

DATA Определяет символическое имя, ссылающееся на адрес резидентной памяти данных.

EQU Назначает символическому имени числовое значение или имя регистра.

IDATA Определяет символическое имя, ссылающееся на косвенно адресуемый адрес резидентной памяти данных.

SEGMENT Объявляет имя перемещаемого сегмента, его тип и расположение.

SET Назначает символическое имя числовому значению или регистру.

Имя может быть впоследствии изменено с помощью директивы SET.

XDATA Определяет символическое имя, ссылающееся на адрес внешней памяти данных.

Директивы резервирования и инициализации памяти

Эти директивы используются для резервирования и инициализации слов, байтов или битов. В абсолютном сегменте зарезервированное пространство начинается с текущего адреса. В перемещаемом сегменте зарезервированное пространство начинается с текущего смещения. Указатель расположения поддерживается отдельно для каждого сегмента, к нему можно обращаться, используя символ ($).

Директива Описание

DB Заносит в память программ байтовую константу.

DBIT Резервирует пространство в битовом сегменте.

DS Резервирует пространство памяти в текущем сегменте.

DW Инициализирует память значением слова.

Директивы компоновки программы

Вы можете использовать директивы компоновки программы для того, чтобы дать объектному модулю имя и определить общие и внешние символы. Эти директивы используются в BL51 для объединения отдельных объектных модулей в единый абсолютный объектный модуль.

Директива Описание

EXTRN Определяет символические имена, которые объявлены в других объектных модулях.

NAME Определяет имя объектного модуля.

PUBLIC Определяет символические имена, которые могут использоваться в других объектных модулях.

Директивы управления состоянием ассемблера Эти директивы используются для того, чтобы сообщить о конце трансляции про- граммы, выбрать начальный адрес или смещение для сегмента, определить используемый банк регистров.

Директива Описание

END Сообщает о конце транслируемого модуля.

ORG Изменяет значение ассемблерного счётчика адреса текущего сегмента про-граммы.

USING Выбирает номер банка регистров общего назначения.

Директивы выбора сегмента

Следующие директивы определяют сегменты данных и кода.

Директива Описание

BSEG Выбирает абсолютный битовый сегмент.

CSEG Выбирает сегмент программы в машинном коде.

DSEG Выбирает абсолютный сегмент резидентной памяти данных.

ISEG Выбирает абсолютный косвенно адресуемый сегмент резидентной памяти данных.

RSEG Выбирает предварительно определенный перемещаемый сегмент.

XSEG Выбирает абсолютный сегмент внешней памяти данных.

Директивы макроопределений

Следующие директивы используются для определения макрокоманд.

Директива Описание

ENDM Заканчивает макроопределение.

EXITM Заставляет макрорасширение немедленно завершиться.

IRP Определяет список аргументов.

IRPC Определяет аргумент.

LOCAL Определяет до 16 локальных символов, используемых внутри макрокоманды.

MACRO Начало макроопределения, определяет имя макрокоманды и парамет- ров, которые могут быть переданы макрокоманде.

REPT Определяет количество повторений последующих строк.

Пример 1. Запись данных.

Записать в резидентную память данных по адресам 41 и 42 число 1C3FH:

```
LOAD: MOV R0,#41H ;загрузка в R0 указателя данных
MOV @R0,#1CH ;запись в память числа 1CH
INC R0 ;инкремент указателя
MOV @R0,#3FH ;запись в память числа 3FH
```

Дополним программу директивами и командами ассемблера, обеспечивающими отладку и тестирование:

```
START: JMP LOAD ;переход к программе
ORG 2100H;директива размещения программы с адреса 2100
LOAD: MOV R0,#41H ;загрузка в R0 указателя данных
MOV @R0,#1CH ;запись в память числа 1CH
INC R0 ;инкремент указателя
MOV @R0,#3FH ;запись в память числа 3FH
JMP LOAD ;зацикливание программы
END ;директива окончания трансляции
```

Пример 2. Сложение.

Сложить два двоичных многобайтных числа. Слагаемые располагаются в резидентной памяти данных, начиная с младшего байта. Начальные адреса слагаемых заданы в R0 и R1, формат слагаемых в байтах - в R2:

```
CLR C ;сброс переноса
LOOP: MOV A, @R0 ;загрузка в A текущего байта первого ;слагаемого
ADDC A, @R1 ;сложение байтов с учетом переноса
MOV @R0, A ;размещение байта результата
INC R0 ;продвижение указателей
INC R1
DJNZ R2,
LOOP ;цикл, если не все байты просуммированы
```

Контрольные вопросы

1. По каким функциональным группам можно классифицировать команды микроконтроллера?

2. Какой формат может иметь команда?

3. Как длительность машинного цикла микроконтроллера соотносится с его тактовой частотой?

4. Как определить время выполнения команды?

5. С какими типами данных может оперировать микроконтроллер?

6. Для чего используются четырёхбитные операнды?

7. Какие команды работают с четырёхбитными операндами?

8. Для чего используются двухбайтные операнды?

9. Как косвенно адресуются байты памяти?

10. Укажите назначение флагов слова состояния программы PSW.

11. Сформулируйте условия установки флага OV.

12. Каково назначение регистров указателей?

13. Может ли порт одновременно являться источником операнда и приемником результата операции?

14. Какие способы адресации используются в микроконтроллере?

Лабораторная работа №5. Способы адресации операндов в командах.

Цель работы: познакомить с понятием *"множество"* в языке программирования Pascal; выработать навыки работы со структурой данных множество.

Общие сведения

Изучение команд пересылки данных – одно из наиболее объемных заданий. Оно предусматривает изучение логики работы каждой из перечис- ленных ранее команд. Кроме изучения логики работы команд пересылки данных, при выпол- нении данного задания надо рассмотреть все режимы адресации на примере одной из команд. Лучше всего для этих целей подходит команда MOV. Также надо будет указать результат каждой операции. Направление пере- сылки можно указать стрелкой. Для каждого из режимов адресации укажи- те в комментарии, какие регистры можно (или какие регистры нельзя) ис- пользовать.

Например,

var a : integer; begin a := 12;

asm // РЕЖИМЫ АДРЕСАЦИИ

// непосредственный режим адресации, в качестве приемника // нельзя использовать регистры (укажите регистры)

MOV AX, 23 // источник – непосредственная, приемник – регистровая, (23=>AX)

MOV AX, a // источник – прямая, приемник – регистровая (a => AX = 12)

//косвенно-регистровая адресация (можно использовать регистры…)

LEA EBX, a // адрес переменной a => EBX (EBX = 00409000h) MOV CX, [EBX]

// из ячейки памяти с адресом

EBX = 00409000h => CX = 12 end;

end.

При использовании команд получения адреса (например, LEA) не забывайте показать, как этот адрес используется (см. следующую за коман- дой LEA строку). При

этом покажите, какие числа пересылаются. При рассмотрении команд LES и LDS обратите внимание на сущест- венное отличие этих команд от команды LEA.

Команда LEA определяет адрес размещения самой переменной языка C++, а команды LES и LDS заносят в регистры содержимое переменной типа указатель. Чтобы не потерять значение сегмента данных (DS), его содержимое сохраняется в стеке (командой PUSH DS), а после выполнения команды LDS восстанавливается из стека (командой POP DS):

```
var p : ^integer;
begin
getmem(p,2);
 p^ = 7;
 ASM LES EBX, p
MOV CX, [EBX] // p^ = 7 => CX
PUSH DS
 LES EBX, p
MOV CX, [EBX]
POP DS
END;
end.
```

При реализации в командах косвенно-регистрового режима адресации со смещением и выполнении команды XLAT, используйте массивы данных с обязательным указанием того, к какому элементу массива идет обращение. Содержимое массивов должно быть подготовлено на языке C++ и указа но в комментарии, что там находится. Например:

```
 var M : array[0..15] of char; C : char; i : byte;
 begin // Заполнить массив M символами 16-ричных цифр от 0 до F C := '0';
 for i := 0 to 9 do
 begin
 M[i] := C;
 inc(C);
 end;
 C := 'A';
```

```
 for i := 10 to 15 do
begin
M[i] := C;
inc(C);
end; // Извлечь из массива символ с номером 13 и поместить в переменную C ASM
MOV AL, 13
LEA EBX, M // 0 1 2 3 4 5 6 7 8 9 A B C D E F XLAT
 // выбирается элемент ---^
MOV C, AL // AL => C = 'D'
END;
end.
```

Примерные варианты контрольных задач:

1. Обменять значения в переменных языка C++

int x и int y.

2. Обменять значения в переменных языка C++ x[4] и y^[3];

при выборке значения из массива y^ используйте команду XLAT.

4. Обменять значения в переменных языка C++ x[4] и y^[3].
5. Используйте команды PUSH и POP для временного хранения значений элементов массива.
6. Определите, сколько байт требуется на запись в оперативной памяти команды LEA EBX,M, и какие числа записаны в этих байтах.
7. Используя команды пересылок, покажите, как работает команда CMC.
8. Содержимое регистра флагов поместите в переменную int x.
9. Обменять значения в переменных языка C++ int x и int y. Обязательно использовать команду XCHG.

Лабораторная работа №6. Память

Цель работы: Сравнение различных способов обхода памяти, программное определение размера и степени ассоциативности кэш-памяти различных уровней.

Общие сведения

Кэш-память является промежуточным хранилищем данных между процессором и оперативной памятью. Она содержит копии наиболее часто используемых блоков данных из оперативной памяти. Размер кэш-памяти составляет от нескольких килобайт до нескольких мегабайт, а скорость доступа к ней в несколько раз превосходит скорость доступа к оперативной памяти, но уступает скорости обращения к регистрам. Каждый раз, когда к ячейке оперативной памяти происходит обращение (чтение или запись), ее копия заносится в кэш-память, вытеснив при этом оттуда копию другой ячейки. Поэтому повторное обращение к той же ячейке произойдет быстрее. Значения переменных программы и небольшие массивы, для которых не нашлось места в регистрах, обычно располагаются в кэш-памяти. Большие массивы могут поместиться в кэш-память только частично. Допустим, некоторая программа производит многократную обработку элементов массива. Если построить график зависимости времени обработки массива от размера массива, то он должен иметь нелинейный характер. При превышении размера кэш-памяти время обращения к элементам массива несколько возрастет (на графике будет наблюдаться скачок). Данные из оперативной памяти в кэш-память (и обратно) считываются целыми строками. Размер кэш-строки в большинстве распространенных процессоров составляет 16, 32, 64, 128 байт. При последовательном обходе попытка чтения первого элемента кэш-строки вызывает копирование всей строки из медленной оперативной памяти в кэш. Чтение нескольких последующих элементов выполняется намного быстрее, т.к. они уже находятся в быстрой кэш-памяти. В большинстве современных микропроцессорах реализована аппаратная предвыборка данных. Ее суть состоит в том, что при последовательном обходе очередные кэш-строки копируются из оперативной памяти в кэш-память еще до того, как к ним произошло обращение. За счет этого скорость последовательного обхода данных еще возрастает.

Большинство современных микропроцессоров имеют множественно-ассоциативную (наборно-ассоциативную) организацию кэш-памяти. При множественно-

ассоциативной организации кэш-память разделена на несколько банков, и каждый блок данных из оперативной памяти может быть помещен в одну из определенного множества (набора) строк кэш-памяти. Число строк в множестве определяется числом банков. Схема кэш-памяти данных первого уровня на Pentium III (16 Кб):

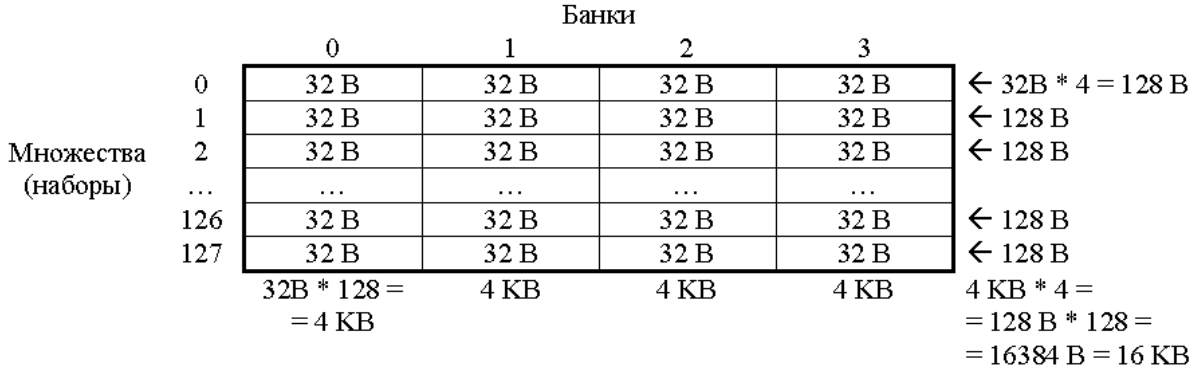

Рисунок 5.

В какой конкретный элемент множества строка будет записана, определяется алгоритмом замещения (циклический, случайный, LRU, псевдо-LRU, …). Таким образом, блоки, отстоящие на определенное расстояние в памяти (в примере: на 212 B = 4096 B = 4 KB), помещаются в одно и то же множество строк. Число элементов в каждом множестве (число банков) называется степенью ассоциативности кэш-памяти. Например, кэш данных L1 в Pentium III имеет объем 16 KB, степень ассоциативности 4 (4-way set-associative), размер строки 32B:

16KB = 4-way * 4 KB = 4-way * 128 множеств * 32B

Данные, расположенные в памяти с шагом на расстоянии 4KB приходятся на одно множество. На все эти данные приходится всего 4 кэш-строки, т.е. 4 * 32 = 128 B. Если выполнять обход данных с шагом 4 KB, то из всех 16 KB кэша L1 будет использоваться всего 128 B, которые будут постоянно перезаписываться (эффект «буксования» кэша). Производительность при этом будет такая же, как при отсутствии кэш-памяти. Если вычислительная система имеет несколько уровней кэш-памяти, то у каждого уровня может быть своя степень ассоциативности. Определить степени ассоциативности кэш-памяти можно следующим способом. Выполняется обход N блоков данных суммарным объемом BlockSize, отстоящих друг от друга на величину Offset:

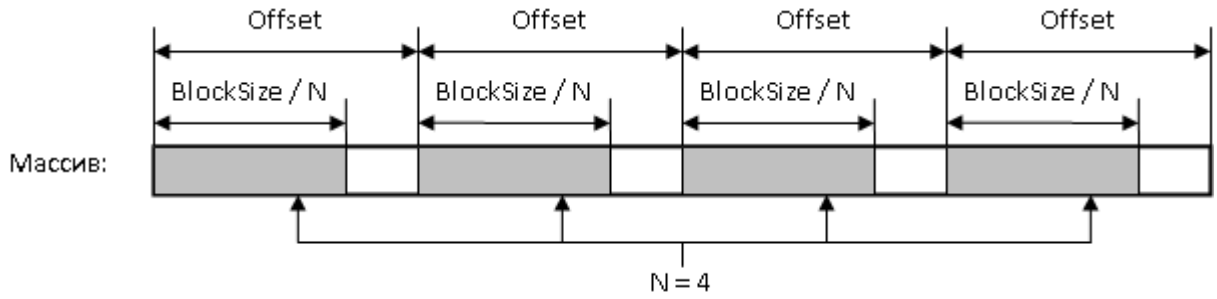

Рисунок 6.

BlockSize должен быть не больше объема исследуемого уровня кэш-памяти. Offset должно быть кратно величине «размер кэша» / «ассоциативность», т.е. кратно размеру банка ассоциативности. Как правило, это степени двоек, так что можно взять заведомо кратное такому значению расстояние (например, 1MB). Изменяя число частей N, мы увидим, как меняется время обхода. Когда N превысит число банков, время обхода сильно возрастет.

Обход элементов следует производить в таком порядке:

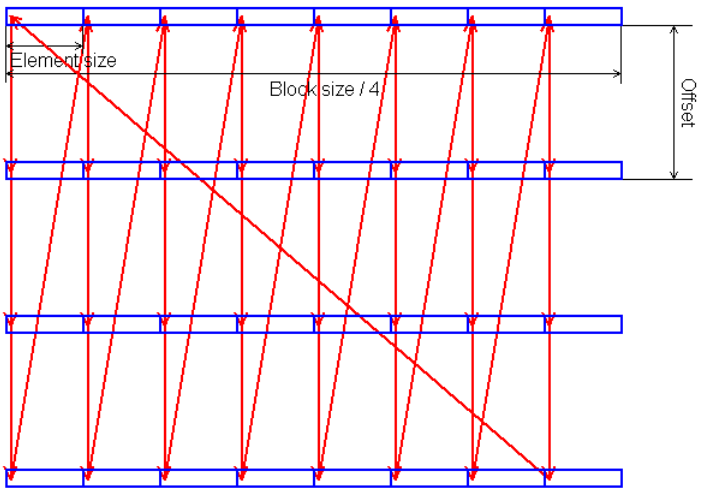

Рисунок 7.

2. Функции замера времени

Для замера времени небольших операций (несколько сотен инструкций) используется инструкция процессора rdtsc, которая лежит в основе функций WinAPI QueryPerformanceFrequency и QueryPerformanceCounter. Пример использования этих функций приведён ниже:

```
#include <stdio.h>
#include <windows.h>
int main()
{
	LARGE_INTEGER b_start,b_stop,b_time,freq;
	double time, pi;
	QueryPerformanceFrequency(&freq);
	QueryPerformanceCounter(&b_start);
	pi = pi_calculate();
	QueryPerformanceCounter(&b_stop);
	b_time.QuadPart = b_stop.QuadPart - b_start.QuadPart;
	printf("Time: %lf sec Pi = %lf\n",
		(double)(b_time.QuadPart)/(double)(freq.QuadPart) ,pi);
	return 0;
}
```

3. Процедура умножения матриц

Самым быстрым способом обхода является прямой последовательный. Это значит, что после обращения в программе к некоторому элементу происходит обращение к элементу, следующему в памяти прямо за ним. Рассмотрим размещение в памяти двумерного массива в программе на языке Си.

float A[N][N];

Известно, что в языке Си массивы располагаются в памяти по строкам (сначала идут элементы первой строки, затем элементы второй строки и т.д.). Значит, в памяти он разместится следующим образом:

…	$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	…	$A_{0,N-1}$	$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	…	$A_{1,N-1}$	…	$A_{N-1,0}$	$A_{N-1,1}$	$A_{N-1,2}$	…	$A_{N-1,N-1}$	…

Получается два варианта перебора элементов массива:

Быстро:	**Медленно:**

for (i=0;i<N;i++)
  for (j=0;j<N;j++) A[i][j]=x;
	for (j=0;j<N;j++)
	  for (i=0;i<N;i++) A[i][j]=x;

Рассмотрим задачу перемножения двух квадратных матриц N×N. Если напрямую запрограммировать известную формулу: $C_{ik} = \sum_{j=0}^{N-1} A_{ij} B_{jk}$, например, на языке Си, получим следующий фрагмент программы:

```
	for (i=0;i<N;i++)
	  for (k=0;k<N;k++)
	    for (j=0;j<N;j++) C[i][k]+=A[i][j]*B[j][k];
```

Заметим, что в этом случае массив A перебирается по строкам, а массив B – по столбцам (смотрим на внутренний цикл). Зная, что массивы в языке Си хранятся по строкам, приходим к выводу, что элементы массива A перебираются последовательно, а элементы массива B – нет. В данном случае порядок обхода массива C практически не важен, поскольку между обращениями к различным его элементам проходит довольно

много времени. Чтобы ускорить программу, нужно, чтобы, по крайней мере, во внутреннем цикле элементы массивов перебирались последовательно. Для этого необходимо либо заранее транспонировать массив B, либо переставить циклы следующим образом:

```
for (i=0;i<N;i++)
  for (j=0;j<N;j++)
    for (k=0;k<N;k++) C[i][k]+=A[i][j]*B[j][k];
```

Задание.

1. Реализовать прямой обход памяти (Обязательное задание – 1 балл).
2. Реализовать обратный обход памяти (Обязательное задание – 1 балл).
3. Реализовать случайный обход памяти (Обязательное задание – 1 балл).
4. Определить степень ассоциативности кэш-памяти (Обязательное задание – 2 балла).
5. Сравнить умножение двух квадратных матриц с использованием стандартного алгоритма и алгоритма с учётом прямого обхода памяти (Дополнительное задание – 5 баллов).

Лабораторная работа №7. Устройства управления.

Цель работы: Научиться управлять режимами работы микроконтроллеров семейства AVR фирмы Atmel. Понимать способы внутренней и внешней синхронизации работы микроконтроллеров семейства AVR фирмы Atmel.

Общие сведения

Упрощенное устройство синхронизации микроконтроллеров семействаAVR представлено на рис. 2.1. Как видно из рисунка, на основе системного тактового сигнала формируются дополнительные сигналы, используемые для тактирования различных модулей и блоков микроконтроллера:

– clk_{CPU}тактовый сигнал центрального процессора, используетсядля тактирования блоков микроконтроллера, отвечающих за работу с ядром микроконтроллера (регистровый файл, память данных). При выключении этого сигнала ЦПУ останавливается, все вычисления прекращаются;

– $clk_{I/O}$ тактовый сигнал подсистемы ввода/вывода, используетсябольшинством периферийных устройств, таких как таймеры/счетчики и интерфейсные модули. Этот сигнал используется такжеподсистемой внешних прерываний, однако ряд внешних прерываний могут генерироваться и при его отсутствии;

– clk_{FLASH}тактовый сигнал для управления FLASH –памятью программ. Как правило, этот сигнал формируется одновременнос тактовым сигналом центрального процессора;

– clk_{ASY}тактовый сигнал асинхронного таймера/счетчика. Тактирование осуществляется непосредственно от внешнего кварцевого резонатора (32768 Гц). Наличие этого сигнала позволяет использовать соответствующий таймер/счетчик в качестве часовреального времени даже при нахождении микроконтроллерав «спящем» режиме;

– clk_{ADC}тактовый сигнал модуля АЦП. Наличие этого тактовогосигнала позволяет осуществлять преобразования при остановленном ЦПУ и подсистеме ввода/вывода. При этом значительноуменьшается уровень помех, генерируемых микроконтроллером,точность преобразования увеличивается.

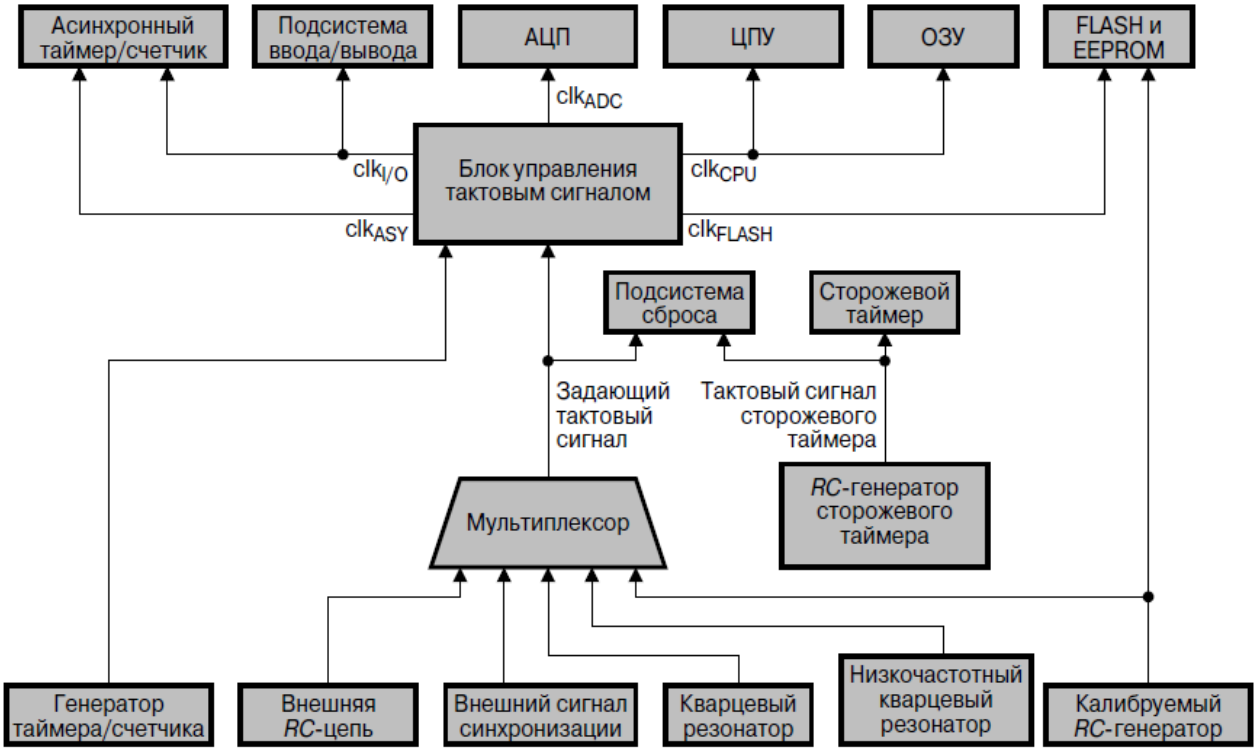

Рис. 2.1. Устройство синхронизации

Тактовый генератор микроконтроллеров семейства AVR может работать с внешним кварцевым резонатором, внешней или внутренней *RC*-цепочкой, а также с внешним сигналом синхронизации. Возможностьиспользования того или иного источника тактового сигнала зависит отмодели микроконтроллера. Поскольку архитектура микроконтроллеров полностью статическая, минимально допустимая частота ничемне ограничена (вплоть до пошагового режима работы), а максимальнаярабочая частота определяется конкретной моделью микроконтроллера.

Выбор режима работы осуществляется программированием конфигурационных ячеек (FUSEBits, FUnctionSElectBits) CKSEL3…0(ClocKSELect) (CKSEL2…0 для ATtiny11x).

Различные модели микроконтроллеров семейства AVR поддерживают от 2 до 6 режимов пониженного энергопотребления (табл. 2.1). Режимы отличаются числом периферийных устройств микроконтроллера, функционирующих в «спящем» режиме и степенью уменьшения энергопотребления.В зависимости от модели для управления «спящим» режимом используется различное число регистров ввода/вывода, которые сведены втабл. 2.2.

Таблица 2.1.Режимы пониженного энергопотребления

Режимы пониженного	Idle	В режиме Idle прекращается формирование тактовых сигналов clk_{CPU} и clk_{FLASH}. При этом ЦПУ микроконтроллера останавливается, а все остальные

энергопотребле-ния		периферийные устройства (интерфейсные модули, таймеры/счетчики, аналоговый компаратор, АЦП, сторожевой таймер), а также подсистема прерываний продолжают функционировать. Поэтому выход из режимаIdle возможен как по внешнему, так и по внутреннему прерыванию. Если разрешена работа АЦП, то преобразование начнет выполняться сразу же после перехода в этот «спящий» режим
	ADC Noise Reduction	Данный режим имеется только в моделях, содержащих в своем составе модуль АЦП. В этом режиме прекращает работу ЦПУ микроконтроллера и подсистема ввода/вывода (отключаются тактовые сигналы clk_{CPU}, clk_{FLASH} и $clk_{I/O}$), а АЦП, подсистема обработки внешних прерываний, сторожевой таймер и блок сравнения адреса модуля TWI продолжают функционировать. За счет этого уменьшаются помехи на входах АЦП, вызываемые работой системы ввода/вывода микроконтроллера, что, в свою очередь, позволяет повысить точность преобразования. Если АЦП включен, преобразование начинается сразу же после перехода в этот «спящий» режим
	Power Down	В режиме Power Down отключаются все внутренние тактовые сигналы, соответственно прекращается функционирование всех систем микроконтроллера, работающих в синхронном режиме. Единственными узлами, продолжающими работать в этом режиме, являются асинхронные модули микроконтроллера: сторожевой таймер (если он включен), подсистема обработки внешних прерываний и блок сравнения адреса модуля TWI. Соответственно выход из режима Power Down возможен либо в результате сброса (аппаратного, от сторожевого таймера, от схемы BOD) или в результате генерации прерываний
	Power Save	Этот режим идентичен режиму Power Down, за одним исключением: если таймер/счетчик микроконтроллера, поддерживающий работу в асинхронном режиме, сконфигурирован для работы в этом режиме, то он будет работать во время «сна» микроконтроллера. Поэтому выход из режима Power Save возможен не только в результате событий, перечисленных при рассмотрении режима Power Down, но и по прерываниям от таймера/счетчика. Разумеется, эти прерывания должны быть разрешены
	Standby	Этот режим доступен только при использовании генератора с внешним резонатором в качестве источника тактового сигнала. Режим Standby полностью идентичен режиму Power Down, за исключением того что тактовый генератор продолжает функционировать. Благодаря этому переход микроконтроллера в рабочий режим происходит гораздо быстрее за 6 машинных циклов
	Extended Standby	Как и режим Standby, этот режим доступен только при использовании генератора с внешним резонатором. Режим Standby полностью идентичен режиму Power Save, за

		исключением того, что тактовый генератор продолжает функционировать. Поэтому интервал между «пробуждением» микроконтроллера и выходом его в рабочий режим составляет всего 6 машинных циклов

Таблица 2.2.Регистры управления «спящими» режимами

Название	Описание	Адрес
MCUCR	Регистр управления микроконтроллером	$35 ($55)
MCUCSR	Регистр управления и состояния микроконтроллера	$34 ($54)
EMCUCR	Дополнительный регистр управления микроконтроллером	$36 ($56)

УКАЗАНИЯ ПО ПРОВЕДЕНИЮ РАБОТЫ

Описание работы устройства управления микроконтроллеров AVR фирмы Atmel представлено в файле «МК AVR.pdf», для его запуска необходимо на Рабочем столе компьютера найти соответствующую иконку и запустить приложение дважды щелкнув левой кнопкой мыши. Перед запуском файла «МК AVR.pdf» в компьютере пользователя должно быть предустановлен приложение AdobeReader версии 8 и выше.

После запуска файла «МК AVR.pdf» появится окно просмотра приложения AdobeReader, причем в левой колонке находится контекстное меню, где указаны основные разделы описания микроконтроллеров семейства AVR фирмы Atmel. В основном поле приложения находится описание, а также полоса прокрутки для перехода к интересующим главам и разделам описания.

ЗАДАНИЕ

Варианты заданий для составления ассемблер-программ в среде AVRStudio 4,предназначенных для работы с аналоговым компаратором представлены в табл. 2.3.

Таблица 2.3.Варианты заданий

№	Описание программы	Вариант
1	Ассемблер-программа должна перевести микроконтроллер в «спящий» режим Idle (предварительно разрешив переход в режим SLEEP), выход из спящего режима инициализировать по прерыванию от внешнего (аппаратного) сигнала	1, 11, 21

2	Ассемблер-программа должна перевести микроконтроллер в «спящий» режим PowerDown(предварительно разрешив переход в режим SLEEP), выход из спящего режима инициализировать по прерыванию от аналогового компаратора	2, 12, 22
3	Ассемблер-программа должна перевести микроконтроллер в «спящий» режим PowerSave(предварительно разрешив переход в режим SLEEP), выход из спящего режима инициализировать по прерыванию от сторожевого таймера	3, 13, 23
4	Ассемблер-программа должна перевести микроконтроллер в «спящий» режим Standby(предварительно разрешив переход в режим SLEEP), выход из спящего режима инициализировать по прерыванию от внешнего (аппаратного) сигнала	4, 14, 24
5	Ассемблер-программа должна перевести микроконтроллер в «спящий» режим ExtendedStandby(предварительно разрешив переход в режим SLEEP), выход из спящего режима инициализировать по прерыванию от внешнего (аппаратного) сигнала	5, 15, 25
6	Ассемблер-программа должна инициализировать микроконтроллер, задав внутренний источник тактового сигнала (*RC*-генератор) частотой 2,0 МГц. Для работы периферийных устройств задать коэффициент предделителя тактового сигнала 32	6, 16
7	Ассемблер-программа должна инициализировать микроконтроллер, задав внутренний источник тактового сигнала (*RC*-генератор) частотой 4,0 МГц. Для работы периферийных устройств задать коэффициент предделителя тактового сигнала 16	7, 17
8	Ассемблер-программа должна инициализировать микроконтроллер, задав внутренний источник тактового сигнала (*RC*-генератор) частотой 1,0 МГц. Для работы периферийных устройств задать коэффициент предделителя тактового сигнала 8	8, 18
9	Ассемблер-программа должна перевести микроконтроллер в «спящий» режим ADCNoiseReduction (предварительно разрешив переход в режим SLEEP), выход из спящего режима инициализировать по прерыванию от АЦП	9, 19
10	Ассемблер-программа должна инициализировать микроконтроллер, задав внешний источник тактового сигнала (*RC*-цепочка). Для работы периферийных	10, 20

	устройств задать коэффициент предделителя тактового сигнала 64	

1) вариант задания определяется номером студента в журнале группы.

2) определив вариант задания, студент должен найти описание устройства управления микроконтроллера семейства AVRи представить его в отчете.

3) открыв файл «МК AVR.pdf», студент должен изучить таблицы и графики, соответствующие описанию устройства управления.

4) изучив структуру и особенности устройства управления, студент должен предварительно составить ассемблер-программу, согласно варианту задания, указанному в табл. 2.3.

5) создать в среде AVRStudio 4 новый файл, название файла составить исходя из фамилии студента на латинице, номера лабораторной работы и его варианта задания (например, 4Ivanov_12).

6) записать составленную ассемблер-программу в среде AVRStudio 4, отладить (с помощью встроенного отладчика Debug) и построить по результатам отладки одноименный проект (в том числе с расширением .asm).

Лабораторная работа №8. Операционные устройства.

Цель работы: Ознакомление с принципом микропрограммного управления.. Изучение методов проектирования операционных устройств (ОУ) ЭВМ. Изучение функционального базиса операционных устройств (ОУ). Приобретение практических навыков в проектировании, отладке и экспериментальном исследовании операционных устройств (ОУ).

Общие сведения

-Целочисленные арифметико-логические устройства: устройства выполнения логических операций, устройства целочисленного сложения/вычитания, устройства целочисленного умножения, устройства целочисленного деления.

- Устройства обработки чисел с плавающей запятой: устройства сложения/вычитания, устройства умножения, устройства деления, устройства вычисления функций.

- Устройства SSE арифметики.

-Операционные устройства состоят из: Регистров для хранения данных; Шин передачи данных; Комбинационных схем реализации функций;

-Накапливающие сумматоры (последовательные).

-Параллельные сумматоры: с последовательным переносом, с параллельным переносом, с условным переносом, с групповой структурой.

ДИРЕКТИВЫ ПРЕПРОЦЕССОРА И ФУНКЦИИ PRINTF() И SCANF()

Цель работы: изучить особенности работы директив препроцессора и функций printf() и scanf(). Директивы препроцессора

Почти все программы на языке С используют специальные команды для компилятора, которые называются директивами. В общем случае директива – это указание компилятору языка С выполнить то или иное действие в момент компиляции программы. Существует строго определенный набор возможных директив, который включает в себя следующие определения: #define, #elif, #else, #endif, #if, #ifdef, #ifndef, #include, #undef.

Директива #define используется для задания констант, ключевых слов, операторов и выражений, используемых в программе. Общий синтаксис данной директивы имеет следующий вид: #define или #define ()

Следует заметить, что символ ‘;’ после директив не ставится. Приведем примеры вариантов использования директивы #define.

Листинг 1. Примеры использования директивы #define.

```
#include #define TWO 2
#define FOUR TWO*TWO
#define PX printf(“X равно %d.\n”, x)
 #define FMT «X равно %d.\n»
#define SQUARE(X) X*X
int main()
{
int x = TWO;
 PX;
x = FOUR;
printf(FMT, x);
 x = SQUARE(3);
PX;
return 0;
```

}

После выполнения этой программы на экране монитора появится три строки:

X равно 2.

X равно 4.

X равно 9.

Директива #undef отменяет определение, введенное ранее директивой #define. Предположим, что на каком-либо участке программы нужно отменить определение константы FOUR. Это достигается следующей командой: #undef FOUR Интересной особенностью данной директивы является возможность переопределения значения ранее введенной константы. Действительно, повторное использование директивы #define для ранее введенной константы FOUR невозможно, т.к. это приведет к сообщению об ошибке в момент компиляции программы. Но если отменить определение константы FOUR с помощью директивы #undef, то появляется возможность повторного использования директивы #define для константы FOUR. Для того чтобы иметь возможность выполнять условную компиляцию, используется группа директив #if, #ifdef, #ifndef, #elif, #else и #endif. Приведенная ниже программа выполняет подключение библиотек в зависимости от установленных констант. 14 #if defined(GRAPH) #include //подключение графической библиотеки #elif defined(TEXT) #include //подключение текстовой библиотеки #else #include //подключение библиотеки ввода-вывода #endif Данная программа работает следующим образом. Если ранее была задана константа с именем GRAPH через директиву #define, то будет подключена графическая библиотека с помощью директивы #include. Если идентификатор GRAPH не определен, но имеется определение TEXT, то будет использоваться библиотека текстового ввода/вывода. Иначе, при отсутствии каких-либо определений, подключается библиотека ввода/вывода. Вместо словосочетания #if defined часто используют сокращенные обозначения #ifdef и #ifndef и выше приведенную программу можно переписать в виде: #ifdef GRAPH #include //подключение графической библиотеки #ifdef TEXT #include //подключение текстовой библиотеки #else #include //подключение библиотеки ввода-вывода #endif Отличие директивы #if от директив #ifdef и #ifndef заключается в возможности проверки более разнообразных условий, а не только существует или нет какие-либо константы. Например, с помощью директивы #if можно проводить такую проверку: #if SIZE == 1 #include // подключение математической библиотеки #elif SIZE > 1 #include // подключение библиотеки обработки // массивов #endif В приведенном примере подключается либо математическая библиотека, либо библиотека обработки массивов, в зависимости от значения константы SIZE. Используемая в приведенных примерах директива #include позволяет добавлять в программу ранее написанные программы и сохраненные в виде файлов. Например, строка #include 15 указывает препроцессору добавить содержимое файла stdio.h вместо приведенной строки. Это дает большую гибкость, легкость программирования и наглядность создаваемого текста программы.

Лабораторная работа №9. Системы ввода-вывода.

Цель работы: Ознакомление с принципом микропрограммного управления.. Изучение методов проектирования периферийными устройствами ЭВМ. Приобретение практических навыков в проектировании, отладке и экспериментальном исследовании периферийных устройств .

Общие сведения

Функция printf() позволяет выводить информацию на экран при программировании в консольном режиме. Данная функция определена в библиотеке stdio.h и имеет следующий синтаксис:

```
int printf( const char *format [, argument]... );
```

Здесь первый аргумент *format определяет строку, которая выводится на экран и может содержать специальные управляющие символы для вывода переменных. Затем, следует список необязательных аргументов, которые поясняются ниже. Функуция возвращает либо число отображенных символов, либо отрицательное число в случае своей некорректной работы.

В самой простой реализации функция printf() просто выводит заданную строку на экран монитора:

```
printf("Привет мир.");
```

Однако с ее помощью можно выводить переменные разного типа: начиная с числовых и заканчивая строковыми. Для выполнения этой операции используются специальные управляющие символы, которые называются спецификаторами и которые начинаются с символа %. Следующий пример демонстрирует вывод целочисленной переменной num на экран монитора с помощью функции printf():

```
int num;
 num = 5;
 printf("%d", num);
```

В первых двух строках данной программы задается переменная с именем num типа int. В третьей строке выполняется вывод переменной на экран. Работа функции printf() выглядит следующим образом. Сначала функция анализирует строку, которую необходимо вывести на экран. В данном случае это «%d». Если в этой строке встречается спецификатор, то на его место записывается значение переменной, которая является вторым аргументом функции printf(). В результате, вместо исходной строки «%d» на экране появится строка «5», т.е. будет выведено число 5.

Следует отметить, что спецификатор «%d» выводит только целочисленные типы переменных, например int. Для вывода других типов следует использовать другие спецификаторы. Ниже перечислены основные виды спецификаторов:

%c – одиночный символ
%d – десятичное целое число со знаком
%f – число с плавающей точкой (десятичное представление)
%s – строка символов (для строковых переменных)
%u – десятичное целое без знака
%% - печать знака процента

С помощью функции printf() можно выводить сразу несколько переменных. Для этого используется следующая конструкция:

```
 int num_i;
 float num_f;
num_i = 5;
num_f = 10.5;
printf("num_i = %d, num_f = %f", num_i, num_f);
```

Результат выполнения программы будет выглядеть так:

num_i = 5, num_f = 10.5

Кроме спецификаторов в функции printf() используются управляющие символы, такие как перевод строки \n, табуляции \t и др. Например, если в ранее рассмотренном примере необходимо вывести значения переменных не в строчку, а в столбик, то необходимо переписать функцию printf() следующим образом:

printf("num_i = %d,\n num_f = %f", num_i, num_f);

Аналогично используется и символ табуляции. Для ввода информации с клавиатуры удобно использовать функцию scanf() библиотеки stdio.h, которая имеет следующий синтаксис:

int scanf(const char *format [,argument]...);

Здесь, как и для функции printf(), переменная *format определяет форматную строку для определения типа вводимых данных и может содержать те же спецификаторы что и функция printf(). Затем, следует список необязательных аргументов. Работа функции scanf() демонстрируется на листинге 2.

Листинг 2. Пример использования функции scanf().

```
#include<stdio.h>
 int main()
{
int age;
float weight;
 printf("Введите информацию о Вашем возрасте: ");
 scanf("%d", &age);
printf("Введите информацию о Вашем весе: ");
scanf("%f", &weigth);
printf("Ваш возраст = %d, Ваш вес = %f", age, weigth);
 return 0;
 }
```

Основным отличием применения функции scanf() от функции printf() является знак & перед именем переменной, в которую записываются результаты ввода.

Функция scanf() может работать сразу с несколькими переменными. Предположим, что необходимо ввести два целых числа с клавиатуры. Формально для этого можно дважды вызвать функцию scanf(), однако лучше воспользоваться такой конструкцией:

scanf(" %d, %d ", &n, &m);

Функция scanf() интерпретирует это так, как будто ожидает, что пользователь введет число, затем – запятую, а затем – второе число. Функция scanf() возвращает число успешно считанных элементов. Если операции считывания не происходило, что бывает в том случае, когда вместо ожидаемого цифрового значения вводится какая-либо буква, то возвращаемое значение равно 0.

Задание на лабораторную работу

1. Написать программу работы с директивами препроцессора в соответствии с номером своего варианта.

2. Написать программу с использованием функций printf() и scanf() в соответствии с номером своего варианта.

3. Сделать выводы о полученных результатах работы программ.

Варианты заданий

Вариант	Программирование директив препроцессора	Программирование функций printf() и scanf()
1	Программа вычисления a+ b с использованием директивы #define	Ввести два вещественных значения и вывести их произведение на экран монитора
2	С помощью директив #if, #else, #elif осуществить выбор строк программы b), либо ab	Ввести два целочисленных значения и вывести их частное на экран монитора
3	Задать константы M и N и вычислить bN)/ MN+(aM	
4	С помощью директивы #define Ввести два целочисленных 18 вычислить 1,2,...,5 значения и вывести их разность на	

Контрольные вопросы

1. Приведите пример использования функции printf() для вывода значений двух целочисленных переменных на экран.

2. Запишите функцию scanf() для ввода символа с клавиатуры

3. Запишите директиву #define для задания константы с именем LENGTH равной 10

4. Приведите пример макроса, позволяющий возводить число в квадрат.

5. С помощью каких директив можно выполнять условную компиляцию программы?

6. Запишите функцию printf() для вывода вещественной переменной с точностью до сотых.