Информация для размещения на официальном сайте ГБПОУ «Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	421
Дата	09.11.2021 г
Время	08-10 – 9-00
Наименование УД/МДК/УП/ПП	МДК 01.01. Системное программирование
Ф.И.О. преподавателя	Сахарчук Т.В.
Электронная почта	saharchyk777@mail.ru
Основная литература	1. Интеллектуальные системы и технологии. Учебник и практикум для СПО Станкевич Л. А. Научная школа: Санкт-Петербургский политехнический университет Петра Великого (г. Санкт-Петербург). Год: 2019 / Гриф УМО СПО https://biblio-online.ru/book/intellektualnye-sistemy-i-tehnologii-445852 2. Федорова Г.Н. Разработка и администрирование баз данных (2-е изд., стер.) учебник«Академия»2017 г. 3. Федорова Г.Н. Информационные системы (6-е изд., стер.) учебник «Академия» 2017г 4. Рудаков А.В. Технология разработки программных продуктов (12-е изд.) учебник«Академия»2018 г. 5. Перлова О.Н. Соадминистрирование баз данных и серверов (1-е изд.) учебник Академия»2018г. 6. Федорова Г.Н. Сопровождение информационных систем (1-е изд.) учебник «Академия» 7. Фёдорова Г.Н. Основы проектирования баз данных (2-е изд., стер.) учебник «Академия»2018г. 8. Основы проектирования приложений баз данных Баженова И.Ю. Интуит НОУ 2016 https://www.book.ru/book/917912 9. Базы данных. (СПО). Учебник Кумскова И.А. КноРус 2019 https://www.book.ru/book/932018 10. Федорова Г.Н. Разработка программных модулей программного обеспечения для компьютерных систем (2-е изд., стер.) учебник «Академия»2017г.
Тема № 61-62	Лабораторная работа на тему: Нестандартные типы данных: структуры и записи
Задание	**Определение переменных программы** Определение переменных является первой и самой главной задачей нашей программы. Для каждой переменной нужно подобрать такой тип, который бы обеспечивал представление в памяти ее значения с достаточным диапазоном и достаточной точностью. *Название*. Во всех фактических значениях этот столбец является символьной строкой из 8 символов. Т.е. для представления его в памяти переменные, в которых хранятся значения этого столбца, должны быть объявлены как: char name1[9], name2[9], name3[9]; дополнительный символ резервируется для обозначения конца символьной строки. *Школа*. Эти данные являются одним символом. Их объявление: char sch1, sch2, sch3;

	Количество. Эти данные являются целыми числами в диапазоне 10 - 220. Этот диапазон перекрывается диапазоном возможных значений типа ***unsigned short*** (диапазон этого типа: 0 - 255). Т.е. объявление соответственных переменных:
	unsigned short cnt1, cnt2, cnt3;
	Это число с дробной частью, следовательно, соответствующие переменные должны быть переменными с плавающей точкой. Точность данных - всего один знак после точки, поэтому для их представления будет достаточно типа ***float***:
	float sq1, sq2, sq3;
	Разработка текста программы
	Начинаем разработку текста програмы с заголовка главной функции ***main():***
	int main(void)
	Далее открывается тело функции и в нее включается описание переменных.
	Кодовая часть программы начинается с приглашения - вывода строки-константы:
	printf("1. Введите: название, школу, количество, площадь >");
	за которым считываются данные, вводимые оператором:
	scanf("%s %c %d %f",name1, &sc1, &cnt1, &sq1);
	Поскольку описания этих функций хранятся в файле ***stdio.h***, включаем этот файл в начало программы:
	#include <stdio.h>
	При вводе строка вводится по спецификации типа ***%s***, один символ - по спецификации типа ***%c***, целое число - по спецификации типа ***%d***, число с плавающей точкой - по спецификации типа ***%f***. Спецификации разделяются пробелами, т.е. и данные при вводе должны разделятся пробелами или переходом на новую строку. В списке ввода перед всеми элементами кроме того, который вводится по ***%s***, ставится знак ***&.***
	Приглашение-ввод повторяется трижды, с разными переменными в списке ввода.
	Для формирования вывода следует подсчитать ширину каждого столбца. Ширина первого столбца определяется размером фактических данных в строке - 9 символьных мест, с учетом пробелов в начале и в конце - 11. Ширина остальных столбцов определяется шириной текста в заголовках столбцов и составляет соответственно 7, 11 и 13 символьных мест. Учитывая вертикальные линии ширина строки составляет 47 знакомест.
	Первая строка - горизонтальная линия, которая состоит из символа '-', повторенного 47 раз. Вторая строка - общий заголовок, дополненный до ширины 47 пробелами. Третья строка - еще одна горизонтальная линия. Четвертая и пятая строки - заголовки столбцов, каждый из них имеет установленную ширину. Шестая строка - еще одна горизонтальная линия.
	Наконец, строки с седьмой до девятой - фактические даные. В каждой строке выводятся значения набора переменных для одной строки. Например:
	printf("| %9s | %c | %-3d | %5.1f |\n", name1, sc1, cnt1, sq1);
	Значения имени выводятся по спецификации типа ***%s*** с шириной 9. Значения школы выводятся по спецификации

типа ***%c***. Поскольку значения в этом столбце выводится по центру столбца, перед и после него ставятся пробелы. Значения количества выводятся по спецификации типа ***%d*** с шириной 3, а площадь - ***%f*** с общей шириной 5 и одним знаком после точки. Значения в этих столбцах дополняются до нужной ширины пробелами. В столбце 3 ставится признак выравнивания по правому краю.

Остальные четыре строки выводятся как текстовые константы.

Полный текст программы приведен ниже.

```
/*********************************************************/
/*          Лабораторная  работа №2                      */
/*          Типы данных и ввод-вывод                     */
/*       Пример выполнения. Вариант №30.                 */
/*********************************************************/
#include <stdio.h>
int main(void) {
 char name1[9], name2[9], name3[9];
 char sc1, sc2, sc3;
 unsigned short cnt1, cnt2, cnt3;
 float sq1, sq2, sq3;
 /* Введение фактических данных*/
 printf("1. Введите: название, школу, количество, площадь >");
 scanf("%s %c %d %f",name1, &sc1, &cnt1, &sq1);
 printf("2. Введите: название, школу, количество, площадь >");
 scanf("%s %c %d %f",name2, &sc2, &cnt2, &sq2);
 printf("3. Введите: название, школу, количество, площадь >");
 scanf("%s %c %d %f",name3, &sc3, &cnt3, &sq3);
 /* Вывод таблицы */
 /* вывод заголовков */
 printf("-----------------------------------------\n");
 printf("|Буддийские монастыри Японии периода Нара |\n");
 printf("|-----------------------------------------|\n");
 printf("| Название | Школа | Количество|Площадь земли|\n");
 printf("|          |       |   монахов|    (га)     |\n");
 printf("|----------|-------|----------|-------------|\n");
 /* вывод строк фактических данных */
 printf("| %9s | %c | %3d | %-5.1f |\n", name1, sc1, cnt1, sq1);
 printf("| %9s | %c | %3d | %-5.1f |\n", name2, sc2, cnt2, sq2);
 printf("| %9s | %c | %3d | %-5.1f |\n", name3, sc3, cnt3, sq3);
 /* вывод примечаний */
 printf("|-----------------------------------------|\n");
 printf("| Примечание: Т - Тэндай; С - Сингон;       |\n");
 printf("|             Д - Дзедзицу                  |\n");
 printf("-----------------------------------------\n");
 return 0;
}
```

Отладка программы

При отладке программы можно использовать пошаговый режим отладки с отслеживанием значений переменных - тех, которые вводятся. Если возникнут проблемы с вводом переменных, есть смысл вводить каждую переменную отдельным оператором. Аккуратный формат таблицы достигается несколькими запусками

	программы с последующим выравниванием столбцов по результатам запуска. **Результаты работы программы** При работе программы на экран было выдано следующее: 1. Введите: название, школу, количество, площадь > Тодайдзи Т 220 368.8 2. Введите: название, школу, количество, площадь > Якусидзи С 50 54.7 3. Введите: название, школу, количество, площадь > Дайаедзи Д 10 12.2 ----------

```
|Буддийские монастыри Японии периода Нара    |
|--------------------------------------------|
| Название | Школа | Количество|Площадь земли|
|          |       |  монахов  |    (га)     |
|----------|-------|-----------|-------------|
| Тодайдзи |   Т   |       220 | 368.8       |
| Якусидзи |   С   |        50 |  54.7       |
| Дайаедзи |   Д   |        10 |  12.2       |
|--------------------------------------------|
| Примечание: Т - Тендай; С - Сингон;        |
|           Д - Дзедзицу                     |
----------------------------------------------
```

	Выводы При выполнении лабораторной работы изучены вопросы: • типы данных в языке С, объявление переменных в программе • ввод и вывод данных, форматизации вывода.
Контрольный тест	Ответьте на вопросы: 1. Как определить переменные в программе? 2. С какой функции начинается разработка текста программы? 3. С чего начинается кодовая часть программы? 4. Что используется при отладке программы? 5. Какие типы данных используются в языке С? 6. Как можно объявить переменные в программе? 7. Как можно вводить и выводить данные?

Дата____09.11.2021 г _____ _____ _____

Подпись Ф.И.О. преподавателя

Информация для размещения на официальном сайте ГБПОУ «Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	421
Дата	09.11.2021 г
Время	9-10-10-00
Наименование УД/МДК/УП/ПП	МДК 01.01. Системное программирование
Ф.И.О. преподавателя	Сахарчук Т.В.
Электронная почта	saharchyk777@mail.ru
Основная литература	1. Интеллектуальные системы и технологии. Учебник и практикум для СПО Станкевич Л. А. Научная школа: Санкт-Петербургский политехнический университет Петра Великого (г. Санкт-Петербург). Год: 2019 / Гриф УМО СПО https://biblio-online.ru/book/intellektualnye-sistemy-i-tehnologii-445852 2. Федорова Г.Н. Разработка и администрирование баз данных (2-е изд., стер.) учебник«Академия»2017 г. 3. Федорова Г.Н. Информационные системы (6-е изд., стер.) учебник «Академия» 2017г 4. Рудаков А.В. Технология разработки программных продуктов (12-е изд.) учебник«Академия»2018 г. 5. Перлова О.Н. Соадминистрирование баз данных и серверов (1-е изд.) учебник Академия»2018г. 6. Федорова Г.Н. Сопровождение информационных систем (1-е изд.) учебник «Академия» 7. Фёдорова Г.Н. Основы проектирования баз данных (2-е изд., стер.) учебник «Академия»2018г. 8. Основы проектирования приложений баз данных Баженова И.Ю. Интуит НОУ 2016 https://www.book.ru/book/917912 9. Базы данных. (СПО). Учебник Кумскова И.А. КноРус 2019 https://www.book.ru/book/932018 10. Федорова Г.Н. Разработка программных модулей программного обеспечения для компьютерных систем (2-е изд., стер.) учебник «Академия»2017г.
Тема № 63-64	Лекция на тему: Определение данных. Машинная адресация. Определение размера памяти. Специальные средства отладчика.
Задание	Основная *функция* любого процессора, ради которой он и создается, - это выполнение команд. *Система команд*, выполняемых процессором, представляет собой нечто подобное *таблице истинности* логических элементов или таблице режимов работы более сложных логических микросхем. То есть она определяет логику работы процессора и его реакцию на те или иные комбинации внешних событий. Написание программ для *микропроцессорной системы* - важнейший и часто наиболее трудоемкий этап разработки такой системы. А для создания эффективных программ необходимо иметь хотя бы самое общее *представление* о системе команд используемого процессора. Самые компактные и быстрые программы и подпрограммы создаются на языке *Ассемблер*, использование которого без знания системы команд абсолютно невозможно, ведь язык *Ассемблер* представляет собой символьную *запись* цифровых кодов *машинного языка*, кодов

	команд процессора. Конечно, для разработки программного обеспечения существуют всевозможные *программные средства*. Пользоваться ими обычно можно и без знания системы команд процессора. Чаще всего применяются языки программирования высокого уровня, такие как *Паскаль* и *Си*. Однако *знание* системы команд и языка *Ассемблер* позволяет в несколько раз повысить эффективность некоторых наиболее важных частей программного обеспечения любой *микропроцессорной системы* - от микроконтроллера до персонального компьютера. Мы рассмотрим основные *типы команд*, имеющиеся у большинства процессоров, и особенности их применения. Каждая *команда*, выбираемая (читаемая) из памяти процессором, определяет *алгоритм* поведения процессора на ближайшие несколько тактов. Код команды говорит о том, какую операцию предстоит выполнить процессору и с какими ***операндами*** (то есть кодами данных), где взять исходную информацию для выполнения команды и куда поместить результат (если необходимо). Код команды может занимать от одного до нескольких *байт*, причем *процессор* узнает о том, сколько *байт* команды ему надо читать, из первого прочитанного им байта или слова. В процессоре код команды расшифровывается и преобразуется в набор *микроопераций*, выполняемых отдельными узлами процессора. Но разработчику микропроцессорных систем это *знание* не слишком важно, ему важен только результат выполнения той или иной команды. **Адресация операндов** Большая часть команд процессора работает с кодами данных (*операндами*). Одни команды требуют входных *операндов* (одного или двух), другие выдают выходные *операнды* (чаще один *операнд*). Входные *операнды* называются еще операндами-источниками, а выходные называются операндами-приемниками. Все эти коды *операндов* (входные и выходные) должны где-то располагаться. Они могут находиться во внутренних регистрах процессора (наиболее удобный и быстрый вариант). Они могут располагаться в системной памяти (самый распространенный вариант). Наконец, они могут находиться в устройствах ввода/вывода (наиболее редкий случай). *Определение* места положения *операндов* производится кодом команды. Причем существуют разные методы, с помощью которых процессор может определить, откуда брать входной *операнд* и куда помещать выходной *операнд*. Эти методы называются ***методами адресации*** . Эффективность выбранных *методов адресации* во многом определяет эффективность работы всего процессора в целом. **Методы адресации** Количество *методов адресации* в различных процессорах может быть от 4 до 16. Рассмотрим несколько типичных *методов адресации операндов*, используемых сейчас в большинстве микропроцессоров. **Непосредственная адресация** предполагает, что *операнд* (входной) находится в памяти непосредственно за

кодом команды. *Операнд* обычно представляет собой константу, которую надо куда-то переслать, к чему-то прибавить и т.д. Например, команда может состоять в том, чтобы прибавить число 6 к содержимому какого-то внутреннего регистра процессора. Это число 6 будет располагаться в памяти, внутри программы в адресе, следующем за кодом данной команды сложения.

Рис. 1. Непосредственная адресация.

Прямая (она же абсолютная) адресация предполагает, что *операнд* (входной или выходной) находится в памяти по адресу, код которого находится внутри программы сразу же за кодом команды. Например, команда может состоять в том, чтобы очистить (сделать нулевым) содержимое ячейки памяти с адресом 1000000. Код этого адреса 1000000 будет располагаться в памяти, внутри программы в следующем адресе за кодом данной команды очистки.

Рис.2. Прямая адресация.

Регистровая адресация предполагает, что *операнд* (входной или выходной) находится во внутреннем регистре процессора. Например, команда может состоять в том, чтобы переслать число из нулевого регистра в первый. Номера обоих регистров (0 и 1) будут определяться кодом *команды пересылки*.

Косвенно-регистровая (она же косвенная) адресация предполагает, что во внутреннем регистре процессора находится не сам *операнд*, а его адрес в памяти Например, команда может состоять в том, чтобы очистить ячейку памяти с адресом, находящимся в нулевом регистре. Номер этого регистра (0) будет определяться кодом команды очистки.

Рис. 3.Регистровая адресация.

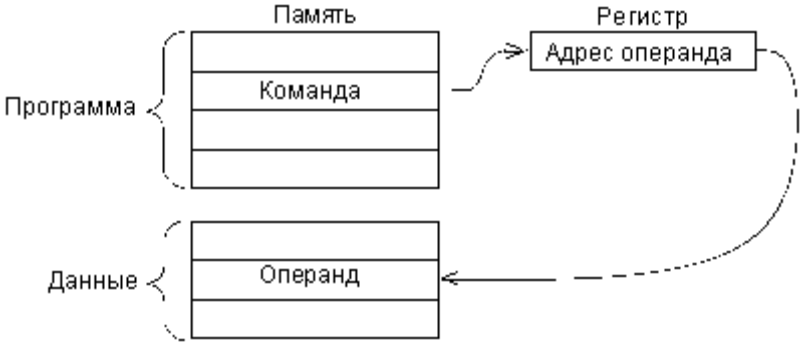


Рис. 4. Косвенная адресация.

Реже встречаются еще два *метода адресации*.

Автоинкрементная адресация очень близка к косвенной адресации, но отличается от нее тем, что после выполнения команды содержимое используемого регистра увеличивается на единицу или на два. Этот *метод адресации* очень удобен, например, при последовательной обработке кодов из массива данных, находящегося в памяти. После обработки какого-то кода адрес в регистре будет указывать уже на следующий код из массива. При использовании косвенной адресации в данном случае пришлось бы увеличивать содержимое этого регистра отдельной командой.

Автодекрементная адресация работает похоже на автоинкрементную, но только содержимое выбранного регистра уменьшается на единицу или на два перед выполнением команды. Эта адресация также удобна при обработке массивов данных. Совместное использование автоинкрементной и автодекрементной адресаций позволяет организовать память стекового типа.

Из других распространенных *методов адресации* можно упомянуть об индексных методах, которые предполагают для вычисления адреса *операнда* прибавление к содержимому регистра заданной константы (индекса). Код этой константы располагается в памяти непосредственно за кодом команды.

Отметим, что выбор того или иного *метода адресации* в значительной степени определяет время выполнения команды. Самая быстрая адресация — это регистровая, так как она не требует дополнительных циклов обмена по магистрали. Если же адресация требует обращения к памяти, то время выполнения команды будет увеличиваться за счет длительности необходимых циклов обращения к памяти. Понятно, что чем больше внутренних регистров у процессора, тем чаще и свободнее можно применять регистровую адресацию, и тем быстрее будет работать система в целом.

Сегментирование памяти

Говоря об адресации, нельзя обойти вопрос о *сегментировании* памяти, применяемой в некоторых процессорах, например в процессорах IBM PC-совместимых персональных компьютеров.

В процессоре Intel 8086 *сегментирование* памяти организовано

следующим образом.

Вся память системы представляется не в виде непрерывного пространства, а в виде нескольких кусков — сегментов заданного размера (по 64 Кбайта), положение которых в пространстве памяти можно изменять программным путем.

Для хранения кодов адресов памяти используются не отдельные регистры, а пары регистров:

- сегментный регистр определяет адрес начала сегмента (то есть положение сегмента в памяти);
- ***регистр указателя*** (регистр смещения) определяет положение рабочего адреса внутри сегмента.

При этом физический 20-разрядный адрес памяти, выставляемый на внешнюю *шину адреса*, то есть путем сложения смещения и адреса сегмента со сдвигом на 4 бита. Сегмент может начинаться только на 16-байтной границе памяти (так как адрес начала сегмента, по сути, имеет четыре младших нулевых разряда, то есть с адреса, кратного 16. Эти допустимые границы сегментов называются границами параграфов.

Отметим, что введение *сегментирования*, прежде всего, связано с тем, что внутренние регистры процессора 16-разрядные, а физический адрес памяти 20-разрядный (16-разрядный адрес позволяет использовать память только в 64 Кбайт, что явно недостаточно). В появившемся в то же время процессоре MC68000 фирмы Motorola внутренние регистры 32-разрядные, поэтому там проблемы *сегментирования* памяти не возникает.

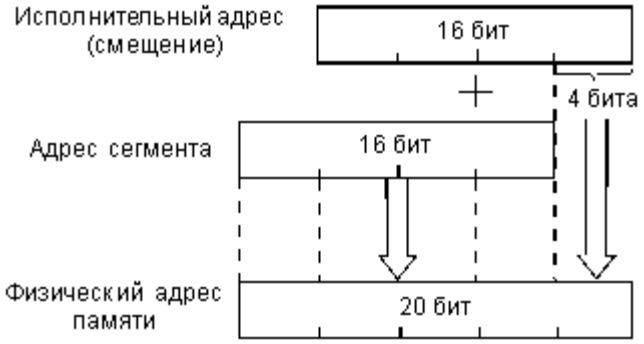


Рис.5. Формирование физического адреса памяти из адреса сегмента и смещения.

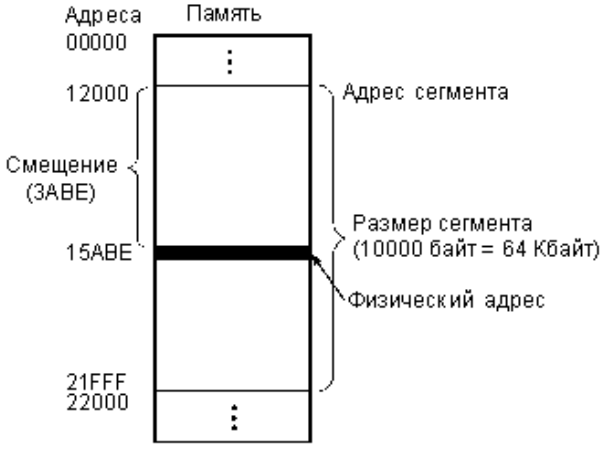


Рис.6. Физический адрес в сегменте (все коды -

шестнадцатеричные).

Применяются и более сложные методы *сегментирования* памяти. В *сегментном регистре* в данном случае хранится не базовый (начальный) адрес сегментов, а коды селекторов, определяющие адреса в памяти, по которым хранятся дескрипторы (то есть описатели) сегментов. Область памяти с дескрипторами называется таблицей дескрипторов. Каждый дескриптор сегмента содержит базовый адрес сегмента, размер сегмента (от 1 до 64 Кбайт) и его атрибуты. Базовый адрес сегмента имеет разрядность 24 бит, что обеспечивает адресацию 16 Мбайт физической памяти.

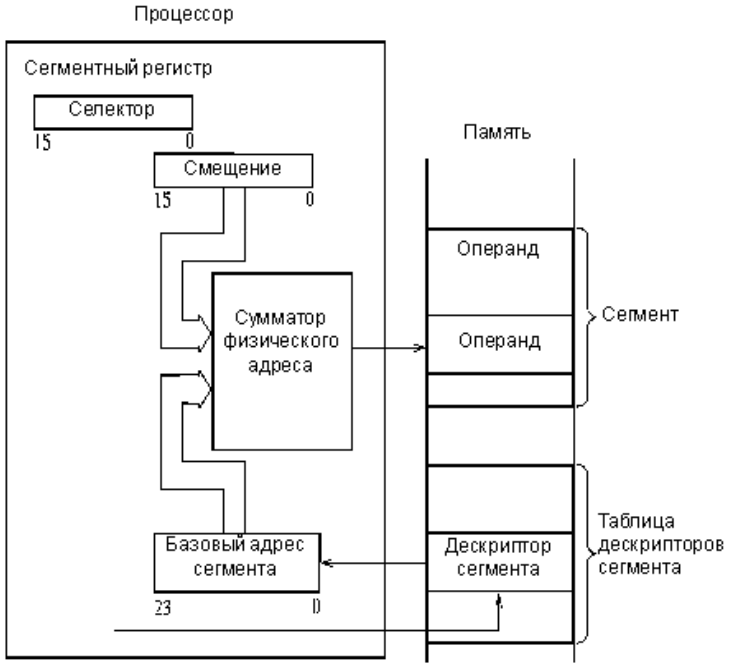


Рис.7. Адресация памяти в защищенном режиме процессора Intel 80286.

Таким образом, на *сумматор*, вычисляющий физический адрес памяти, подается не содержимое сегментного регистра, как в предыдущем случае, а базовый адрес сегмента из таблицы дескрипторов.

Еще более сложный *метод адресации* памяти с *сегментированием* использован в процессоре *Intel 80386* и в более поздних моделях процессоров фирмы Intel. Адрес памяти (физический адрес) вычисляется в три этапа. Сначала вычисляется так называемый **эффективный адрес** (32-разрядный) путем суммирования трех компонентов: базы, индекса и смещения (Base, Index, *Displacement*), причем возможно умножение индекса на масштаб (Scale). Эти компоненты имеют следующий смысл:

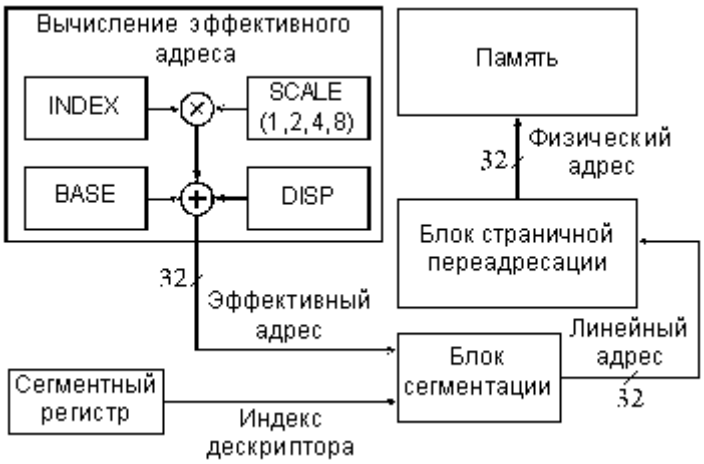


Рис.8. Формирование физического адреса памяти процессора 80386 в защищенном режиме.

- смещение - это 8-, 16- или 32-разрядное число, включенное в команду.
- база - это содержимое базового регистра процессора. Обычно оно используется для указания на начало некоторого массива.
- индекс - это содержимое индексного регистра процессора. Обычно оно используется для выбора одного из элементов массива.
- масштаб - это множитель (он может быть равен 1, 2, 4 или 8), указанный в коде команды, на который перед суммированием с другими компонентами умножается индекс. Он используется для указания размера элемента массива.

Затем специальный блок *сегментации* вычисляет 32-разрядный линейный адрес, который представляет собой сумму базового адреса сегмента из сегментного регистра с эффективным адресом. Наконец, физический 32-битный адрес памяти образуется путем преобразования линейного адреса блоком страничной переадресации, который осуществляет перевод линейного адреса в *физический страницами* по 4 Кбайта.

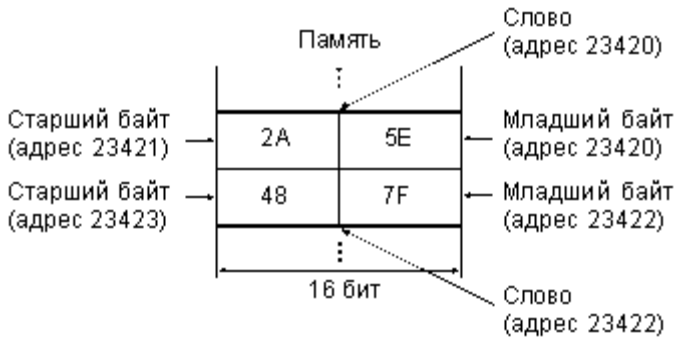
В любом случае *сегментирование* позволяет выделить в памяти один или несколько сегментов для данных и один или несколько сегментов для программ. Переход от одного сегмента к другому сводится всего лишь к изменению содержимого сегментного регистра. Иногда это бывает очень удобно. Но для программиста работать с сегментированной памятью обычно сложнее, чем с непрерывной, несегментированной памятью, так как приходится следить за границами сегментов, за их описанием, переключением и т.д.

Адресация байтов и слов

Многие процессоры, имеющие разрядность 16 или 32, способны адресовать не только целое слово в памяти (16-разрядное или 32-разрядное), но и отдельные байты. Каждому байту в каждом слове при этом отводится свой адрес.

Так, в случае 16-разрядных процессоров все слова в памяти (16-разрядные) имеют четные адреса. А байты, входящие в эти слова, могут иметь как четные адреса, так и нечетные.

Например, пусть 16-разрядная ячейка памяти имеет адрес 23420,

	и в ней хранится код 2A5E  Рис. 9. Адресация слов и байтов. При обращении к целому слову (с содержимым 2A5E) процессор выставляет адрес 23420. При обращении к младшему байту этой ячейки (с содержимым 5E) процессор выставляет тот же самый адрес 23420, но использует команду, адресующую байт, а не слово. При обращении к старшему байту этой же ячейки (с содержимым 2A) процессор выставляет адрес 23421 и использует команду, адресующую байт. Следующая по порядку 16-разрядная ячейка памяти с содержимым 487F будет иметь адрес 23422, то есть опять же четный. Ее байты будут иметь адреса 23422 и 23423. Для различия байтовых и словных циклов обмена на магистрали в *шине управления* предусматривается специальный сигнал байтового обмена. Для работы с байтами в систему команд процессора вводятся специальные команды или предусматриваются методы байтовой адресации.
Контрольный тест	Ответьте на вопросы: 1. Какова основная функция любого процессора? 2. Что такое система команд? 3. Что представляет собой язык Ассемблер? 4. Что представляет собой Адресация операндов? 5. Перечислите методы адресации? 6. Охарактеризуйте автоинкрементную адресацию. 7. Охарактеризуйте автодекрементную адресацию 8. Что такое сегментирование памяти? 9. Что такое адресация байтов и слов?

Дата____09.11.2021 г _____ _____ _____

Подпись Ф.И.О. преподавателя

Информация для размещения на официальном сайте ГБПОУ «Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	421
Дата	10.11.2021 г
Время	8-10-9-00
Наименование УД/МДК/УП/ПП	МДК 01.01. Системное программирование
Ф.И.О. преподавателя	Сахарчук Т.В.
Электронная почта	saharchyk777@mail.ru
Основная литература	1. Интеллектуальные системы и технологии. Учебник и практикум для СПО Станкевич Л. А. Научная школа: Санкт-Петербургский политехнический университет Петра Великого (г. Санкт-Петербург). Год: 2019 / Гриф УМО СПО https://biblio-online.ru/book/intellektualnye-sistemy-i-tehnologii-445852 2. Федорова Г.Н. Разработка и администрирование баз данных (2-е изд., стер.) учебник«Академия»2017 г. 3. Федорова Г.Н. Информационные системы (6-е изд., стер.) учебник «Академия» 2017г 4. Рудаков А.В. Технология разработки программных продуктов (12-е изд.) учебник«Академия»2018 г. 5. Перлова О.Н. Соадминистрирование баз данных и серверов (1-е изд.) учебник Академия»2018г. 6. Федорова Г.Н. Сопровождение информационных систем (1-е изд.) учебник «Академия» 7. Фёдорова Г.Н. Основы проектирования баз данных (2-е изд., стер.) учебник «Академия»2018г. 8. Основы проектирования приложений баз данных Баженова И.Ю. Интуит НОУ 2016 https://www.book.ru/book/917912 9. Базы данных. (СПО). Учебник Кумскова И.А. КноРус 2019 https://www.book.ru/book/932018 10. Федорова Г.Н. Разработка программных модулей программного обеспечения для компьютерных систем (2-е изд., стер.) учебник «Академия»2017г.
Тема № 65-66	Лекция на тему: Организация подпрограмм. Директивы описания подпрограмм, передача параметров в подпрограммы. Команды передачи управления: команды безусловных, условных переходов, команды перехода по соотношению между числами.
Задание	**Подпрограммы** Вначале языки программирования были проще, они выполнялись строго сверху-вниз, один оператор за другим. Такие языки еще называли **линейными**. Типичный пример линейных языков - Бейсик. Единственную возможность организовать хоть какую-то логику в таких языках предоставлял *оператор безусловного перехода* GOTO, который в зависимости от условия, "перепрыгивал" на заранее расставленные метки. В современных языках программирования GOTO тоже остался, наверное, для любителей антиквариата. Но его применение может привести к трудно обнаруживаемым логическим ошибкам времени выполнения (run-time errors). Использование GOTO в современном программировании считается дурным тоном. Мы не будем изучать эту возможность, поскольку для организации

логики есть куда более "продвинутые" средства! Одним из таких средств являются **подпрограммы**.

Подпрограмма - это часть кода, которую можно вызвать из любого места программы неопределенное количество раз.

Другими словами, подпрограммы подобны строительным кирпичикам, из которых, в конце концов, получается здание - *программа*. Без подпрограмм можно обойтись, если вы пишете небольшую учебную программу на пару десятков строк кода. А если это серьезное *приложение*, с парой сотен модулей, в каждом из которых могут быть тысячи строк кода? Как такую программу написать, не разбивая задачу на отдельные части? Подпрограммы помогают улучшить код, структурировать его. Поэтому языки высокого уровня, которые позволяют использовать подпрограммы, называют ещё **процедурно-ориентированными** языками. И наш *компилятор* FPC тоже относится к таким языкам.

Подпрограммы бывают двух типов: **процедуры** и **функции** - первые просто выполняют свою работу, вторые еще и возвращают результат этой работы.

Процедуры

На самом деле, мы уже неоднократно использовали процедуры. Например, когда генерировали событие нажатия на кнопку. Это событие - процедура. Процедура начинается с ключевого слова procedure и имеет следующий *синтаксис*:

```
procedure <имя процедуры>(<список параметров>);
const
  <объявление констант>;
type
  <объявление новых типов>;
var
  <объявление переменных>;
<описание вложенных процедур и функций>;
begin
 <тело процедуры>;
end;
```

Большая часть указанного синтаксиса является необязательной - процедура может не иметь параметров, констант, пользовательских типов данных, переменных и т.п. - она может быть очень простой, например:

```
procedure ErrorMessage;
begin
  ShowMessage('Ошибка!' +#13 + 'На ноль делить нельзя!');
end;
```

Такую процедуру можно вызвать из любого места программы, но процедура обязательно должна быть описана выше - ведь иначе *компилятор* не будет знать о ней. Есть еще возможность предварительно объявить процедуру, но об этом чуть позже. Итак, если эта процедура описана выше, то мы можем вызвать её, просто указав её имя:

```
ErrorMessage;
```

Компилятор перейдет к процедуре и выполнит её код (в данном случае - выведет *сообщение об ошибке*). После

этого *компилятор* вернется назад, и выполнит следующий за вызовом процедуры оператор.

Параметры

Параметры подпрограмм - достаточно важная тема, поговорим об этом подробней. В процедуру (или в функцию) можно передать какие то исходные данные, чтобы процедура их обработала. Такие данные называются **параметрами**, или **формальными параметрами**. Пример - *пользователь* ввел какое-то число (а на самом деле, строку из цифровых символов), нам нужно удвоить его, а результат сообщить пользователю. Описать подобную процедуру можно следующим образом:

```
procedure Udvoenie(st: string);
var
  r: real;
begin
   //полученную строку преобразуем в число:
   r:= StrToFloat(st);
   //теперь удвоим его:
   r:= r * 2;
   //теперь выведем результат в сообщении:
   ShowMessage(FloatToStr(r));
end;
```

Этот пример уже сложнее, правда? На самом деле, всё просто. Давайте разберем, что тут к чему. Итак, строка объявления процедуры:

```
procedure Udvoenie(st: string);
```

объявляет процедуру Udvoenie с параметром строкового типа st. Это означает, что теперь мы можем вызвать процедуру, передав ей в качестве параметра какую-то строку. *Параметр* st условно можно считать внутренней переменной процедуры, в которую *компилятор* скопирует передаваемую процедуре строку. Этот способ передачи данных в подпрограмму называется **параметром по значению**. Допустим, в дальнейшем мы вызвали процедуру таким образом:

```
Udvoenie('123.4');
```

Компилятор сделает вызов процедуры, передав в *параметр* st указанное *значение* '123.4'. Или же мы можем вызвать процедуру иначе, передав в неё *значение*, которое хранится в какой то другой строковой переменной:

```
myst:= '123.4';
Udvoenie(myst);
```

Результат будет таким же. Тут важно помнить, что тип передаваемого значения обязательно должен совпадать с типом параметра. Если *параметр* у нас string, то и передавать ему нужно *значение* типа string. *Компилятор* копирует это *значение* в *параметр*. Другими словами, если внутри процедуры мы изменим *значение* параметра st, это никак не отразится на переменной myst, поскольку мы изменим копию данных, а не сами данные.

Пойдем дальше. А дальше мы объявляем вещественную переменную r:

```
var
```

r: real;

Здесь она необходима, ведь нам нужно умножить *значение* параметра на два, поэтому мы вынуждены будем преобразовать строковое *представление* числа в настоящее число - ведь строку на два не умножишь! Результат поместим в r:

```
begin
  //полученную строку преобразуем в число:
  r:= StrToFloat(st);
```

Служебным словом begin мы начинаем *тело процедуры*. Стандартной функцией StrToFloat(st) мы преобразуем строковое *значение* параметра st в число, и присвоим это число переменной r. Далее всё просто:

```
  //теперь удвоим его:
  r:= r * 2;
  //теперь выведем результат в сообщении:
  ShowMessage(FloatToStr(r));
end;
```

Мы удваиваем *значение* r, результат этого помещаем снова в r, затем стандартной функцией FloatToStr(r) преобразуем полученное число в строку, и выводим эту строку в сообщении ShowMessage(). Вот, собственно, и всё.

Теперь мы сможем вызывать эту процедуру, когда необходимо, и передавать ей различные числа в виде строки. А уж процедура сама позаботится обо всех необходимых преобразованиях, об удвоении числа и выводе результатов на экран.

Кстати, сами данные, которые мы передаём в подпрограмму, называются **аргументами** или **фактическими параметрами**. В примере вызова процедуры

```
myst:= '123.4';
Udvoenie(myst);
```

переменная myst - *аргумент*.

В качестве параметров в процедуре можно использовать не одну, а множество переменных. Если они имеют одинаковый тип, то их имена разделяют запятыми, а тип указывается в конце сразу для всех параметров. Например:

```
procedure MyStrings(st1, st2, st3: string);
```

Если параметры имеют разные типы, их разделяют точкой с запятой:

```
procedure MyProc1(st: string; r1:real);
procedure MyProc2(st1, st2, st3:string; r1:real);
```

Однако, разбавим теорию практикой, и поработаем с процедурами на реальном примере. Откройте **Lazarus** с новым проектом. Как всегда, назовем главную форму (свойство Name) fMain, сохраним проект в папку **09-01**, при этом назовем проект, например, **MyPodprog**, а модулю дадим имя **Main**.

В свойстве Caption формы напишем

Примеры работы с подпрограммами

Наша задача: получить от пользователя *вещественное число*, удвоить его, и результат вывести на экран. *Пользователь* может ввести и *целое число*, но процедура обработает его как вещественное (помните о преобразовании типов в прошлой

лекции?), например, если *пользователь* введет 3, то процедура получит 3.0. В результате вычисления получится 6.0, но FloatToStr() *конечные* нули не выводит, так что *пользователь* увидит на экране просто 6.

Ладно, сейчас нужно решить, как получить у пользователя число. Для этого используем *компонент* TEdit, который нам уже знаком по прошлым лекциям. Для начала установим метку TLabel с поясняющим текстом

Введите любое число:

а рядом установим TEdit. Имена у TLabel и TEdit оставим по умолчанию, TEdit будет называться Edit1. Не забудьте очистить у него свойство Text.

Ниже установите простую кнопку **TButton**, в Caption которой напишите текст:

Пример удвоения №1

Разумеется, будут и другие примеры. Подровняйте компоненты, при необходимости измените их размеры. Наша форма должна выглядеть примерно так:

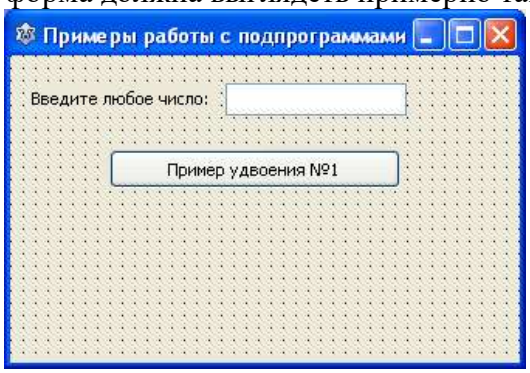


Рис.1. Окно программы MyPodprog

Пока что мы будем вынуждены доверять пользователю, что он введет в *поле* **Edit1** число, и ничего более. Но на прошлой лекции вам было обещано показать реализацию "защиты от дураков", так что чуть позже мы и это сделаем.

Сгенерируйте событие нажатия на кнопку, оно будет таким:

```
procedure TfMain.Button1Click(Sender: TObject);
begin
  Udvoenie(Edit1.Text);
end;
```

Теперь нам нужно создать процедуру Udvoenie **выше** нашего события, сразу после комментария

{ TfMain }

Текст процедуры приведен выше.

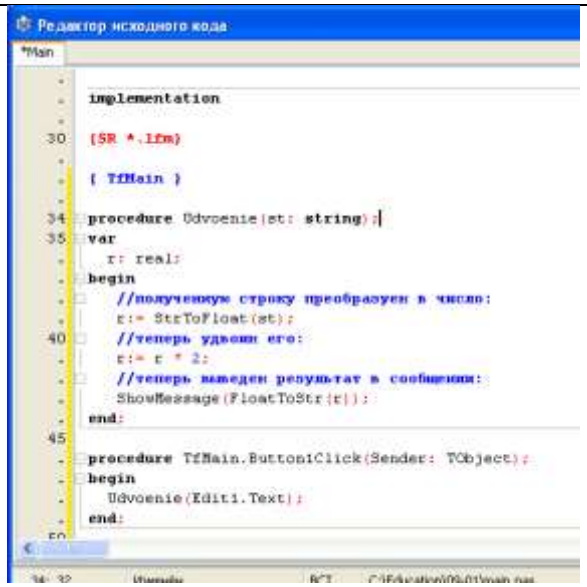
<table>
<tr>
<td></td>
<td>


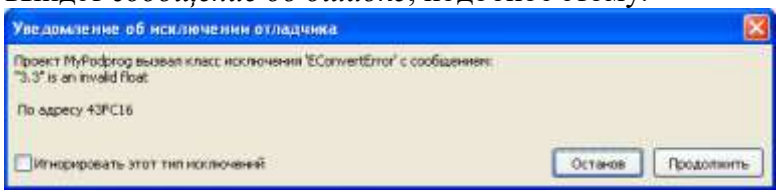
Рис. 2. Реализация подпрограммы Udvoenie
Обратите внимание, мы передаем в подпрограмму значение, которое ввел пользователь, и которое хранится в свойстве Text компонента Edit1:
Udvoenie(Edit1.Text);
Никаких дополнительных переменных в данном случае создавать не нужно. Сохраните проект и запустите его на выполнение. Попробуйте ввести целое число. Затем вещественное. Обратите внимание: если у вас установлена русская версия Windows, то в качестве разделителя вещественного числа нам нужно вводить запятую, а не точку! Помните про глобальную переменную DecimalSeparator?
Если же вы случайно или намеренно ввели точку, то выйдет сообщение об ошибке, подобное этому:


Рис. 3. Сообщение об ошибке
Ничего страшного, нажмите кнопку "Останов", затем выберите команду главного меню "Запуск -> Сбросить отладчик". Lazarus закроет зависший проект, и вы сможете запустить его снова. Похожая ошибка возникнет, если вы попытаетесь удвоить пустую строку. Если же вы ввели числа правильно, то программа отработает как нужно в независимости, целое это было число, или вещественное. Не закрывайте пока проект, он нам еще понадобится.
</td>
</tr>
<tr>
<td>Контрольный тест</td>
<td>
Ответьте на вопросы:

Что такое подпрограмма?
Какие языки называются процедурно-ориентированными?
Каких типов бывают подпрограммы?
Охарактеризуйте процедуры.
Охарактеризуйте функции

</td>
</tr>
</table>

	6. Что такое параметры? 7. Охарактеризуйте параметры по значению 8. Что называется аргументами?

Дата____10.11.2021 г _____ _____ _____

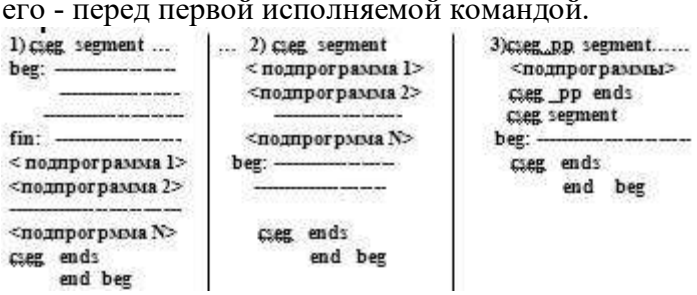
Подпись Ф.И.О. преподавателя

Информация для размещения на официальном сайте ГБПОУ «Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	421
Дата	10.11.2021 г
Время	9-10-10-00
Наименование УД/МДК/УП/ПП	МДК 01.01. Системное программирование
Ф.И.О. преподавателя	Сахарчук Т.В.
Электронная почта	saharchyk777@mail.ru
Основная литература	1. Интеллектуальные системы и технологии. Учебник и практикум для СПО Станкевич Л. А. Научная школа: Санкт-Петербургский политехнический университет Петра Великого (г. Санкт-Петербург). Год: 2019 / Гриф УМО СПО https://biblio-online.ru/book/intellektualnye-sistemy-i-tehnologii-445852 2. Федорова Г.Н. Разработка и администрирование баз данных (2-е изд., стер.) учебник«Академия»2017 г. 3. Федорова Г.Н. Информационные системы (6-е изд., стер.) учебник «Академия» 2017г 4. Рудаков А.В. Технология разработки программных продуктов (12-е изд.) учебник«Академия»2018 г. 5. Перлова О.Н. Соадминистрирование баз данных и серверов (1-е изд.) учебник Академия»2018г. 6. Федорова Г.Н. Сопровождение информационных систем (1-е изд.) учебник «Академия» 7. Фёдорова Г.Н. Основы проектирования баз данных (2-е изд., стер.) учебник «Академия»2018г. 8. Основы проектирования приложений баз данных Баженова И.Ю. Интуит НОУ 2016 https://www.book.ru/book/917912 9. Базы данных. (СПО). Учебник Кумскова И.А. КноРус 2019 https://www.book.ru/book/932018 10. Федорова Г.Н. Разработка программных модулей программного обеспечения для компьютерных систем (2-е изд., стер.) учебник «Академия»2017г.
Тема № 67-68	Лабораторная работа на тему: Передача параметров в подпрограммы: передача параметров через регистры, передача параметров через общие ячейки памяти, передача параметров через стек.
Задание	Программа, оформленная как процедура, к которой обращение происходит из ОС, заканчивается командой возврата ret. Подпрограмма (ПП), как вспомогательный алгоритм, к которому возможно многократное обращение с помощью команды call, тоже оформляется как процедура с помощью директив proc и endp. Структуру процедуры можно оформить так: <имя процедуры> proc <параметры> <тело процедуры> ret <имя процедуры> endp В Ассемблере один тип подпрограмм - процедура. Размещать ее можно в любом месте программы, но так, чтобы управление на нее не попадало случайно, а только по команде call. Поэтому описание ПП принято располагать в конце программного

сегмента (после последней исполняемой команды), или вначале его - перед первой исполняемой командой.



Если программа содержит большое количество подпрограмм, то ПП размещают в отдельном кодовом сегменте - вариант структуры 3).

Замечания:

1) После имени в директивах proc и endp двоеточие не ставится, но имя считается меткой, адресом первой исполняемой команды процедуры.

2) Метки, описанные в ПП, не локализуются в ней, поэтому они должны быть уникальными в рамках всей программы.

3) Параметр в директиве начала процедуры один - FAR или NEAR.

Основная проблема при работе с ПП в Ассемблере - это передача параметров и возврат результатов в вызывающую программу. Существуют различные способы передачи параметров: 1) по значению, 2) по ссылке, 3) по возвращаемому значению, 4) по результату, 5) отложенным вычислением.

Параметры можно передавать: 1) через регистры, 2) в глобальных переменных, 3) через стек, 4) в потоке кода, 5) в блоке параметров.

Передача параметров через регистры - наиболее простой способ. Вызывающая программа записывает в некоторые регистры фактические параметры, подпрограмма использует их для выполнения вспомогательного алгоритма и записывает результат тоже в некоторый регистр. Основная программа использует этот результат. Этот метод используется, если параметров немного. Программист обязан следить за правильностью использования регистров в основной программе и подпрограммах. Примерами использования этого метода являются вызовы некоторых прерываний ОС и BIOS.

Когда регистров не хватает, один из способов обойти это ограничение - записать параметр в глобальную переменную, к которой затем обращаться в ПП. Но этот метод считается не эффективным, так как может оказаться невозможной рекурсия, и даже простое повторное обращение к ПП.

Передача параметров через стек. Сразу перед обращением к процедуре фактические параметры (их значения или адреса) записываются в стек, а процедура их из стека извлекает. Именно этот способ используют языки высокого уровня.

Передача параметров в потоке кода заключается в том, что данные, передаваемые в ПП, располагаются сразу за командой обращения к ПП call. ПП, чтобы использовать эти данные, должна обратиться к ним по адресу, который записывается в стек автоматически как адрес возврата из ПП. Но ПП в этом случае

должна перед командой возврата изменить адрес возврата на адрес байта, следующий за передаваемыми параметрами. Этот метод реализует передачу параметров медленнее, через регистры, глобальные переменные или стек, но примерно так же, как и метод передачи параметров в блоке параметров.
Блок параметров - это участок памяти, содержащий параметры и располагающийся обычно в сегменте данных. Процедура получает адрес начала этого блока при помощи любого из рассмотренных методов: в регистре, в переменной, в стеке, в коде или даже в другом блоке параметров. Примеры использования этого способа - многие функции ОС и BIOS, например, поиск файла, использующий блок параметров DTA, или загрузка и исполнение программы, использующая блок параметров EPB.
Передача параметров по значению.
При передаче параметров по значению процедуре передается значение фактического параметра, оно копируется в ПП, и ПП использует копию, поэтому изменение, модификация параметра оказывается невозможным. Этот механизм используется для передачи параметров небольшого размера.
Например, нужно вычислить c = max (a,b) + max (7, a-1). Здесь все числа знаковые, размером в слово. Используем передачу параметров через регистры. Процедура получает параметры через регистры AX и BX, результат возвращает в регистре AX.
Процедура: AX = max (AX, BX)

```
max proc
cmp AX, BX
jge met1
mov AX, BX
met1: ret
max endp
```

Фрагмент вызывающей программы:

```
---------------------------------
; c = max (a,b) + max (7, a-1)
mov AX, a
mov BX, b
call max ; AX = max (a,b)
mov c, AX ; c = max (a,b)
mov BX, a
Dec BX
call max ;AX = max (7, a-1)
add c, AX
---------------------------------
```

Передача параметров по ссылке.
Оформим как процедуру вычисление x = x div 16
Процедура имеет один параметр-переменную x, которой в теле процедуры присваивается новое значение. Т.е. результат записывается в некоторую ячейку памяти. И чтобы обратиться к процедуре с различными параметрами, например, a и b, ей нужно передавать адреса памяти, где хранятся значения переменных a и b. Передавать адреса можно любым способом, в том числе и через регистры. Можно использовать различные регистры, но чаще используются BX, BP, SI, DI. Пусть адрес параметра передается

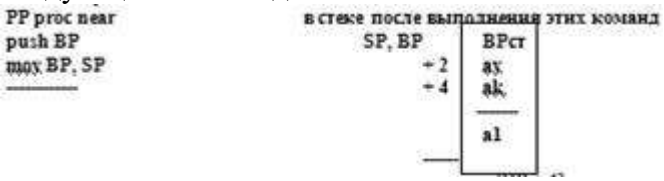
через регистр BX, тогда фрагмент программы:
Фрагмент основной программы:
---
lea BX, a
call Proc_dv
lea BX, b
call Proc_dv

Процедура:
Proc_dv proc
push CX
mov CL, 4
shr word ptr [BX], CL ; x = x div 16
pop CX
ret
Proc_dvendp
Сдвиг на 4 разряда вправо эквивалентен делению нацело на 16 и выполняется быстрее.
Здесь первая команда в процедуре сохраняет в стеке значение регистра CX, так как затем использует CL в команде сдвига и возможно этот регистр используется в основной программе. Т.к. регистров немного а и ПП и основная программа могут использовать одни и те же регистры, то при входе в ПП нужно сохранять в стеке значения регистров, которые будут использоваться в ПП, а перед выходом из нее восстанавливать значения этих регистров. Для поддержки этого, начиная с ix186, в систему команд введены команды сохранения в стеке и извлечения из него сразу всех регистров общего назначения pusa и popa, а, начиная с ix386, pushad popad.
Не нужно сохранять в стеке значение регистра, в который записывается результат работы ПП. Передача параметров по ссылке в блоке параметров
Если параметров много, например, массив, адрес начала массива, как блока параметров, можно передать через регистр, даже если результат ПП не будет записываться по этому адресу.
Даны два массива целых положительных чисел без знака
X DB 100 dup (?)
Y DB 50 dup (?)
Вычислить DL = max (X[i]) + max (Y[i]), использовав процедуру max (A[i]), пересылая адрес массива через регистр BX, а результат сохраняя в AL.
----------; фрагмент программы
lea BX, X
mov CX, 100
call max ; AL = max (X[i])
mov DL, AL ; DL = max (X[i])
lea BX, Y
mov CX, 50
call max ; AL = max (Y[i])
ADD DL, AL

Процедура max: AL = max (A[0..n-1]), BX - начальный адрес A,

CX = n
Max proc
push CX
push BX
mov AL, 0 ; нач. значение max
met1: cmp [BX], AL
jle met2
mov AL, [BX]
met2: inc BX
loop met1
pop BX
pop CX
ret
max endp
Передача параметров через стек.
Этот способ передачи параметров называют универсальным, его можно использовать при любом количестве параметров, хотя он сложнее, чем передача параметров через регистры. Но для передачи результатов чаще используют регистры.
Если ПП имеет k параметров PP(a1, a2, ….ak) размером в слово и параметры сохраняются в стеке в последовательности слева направо, то команды, реализующие обращение к ПП, должны быть следующими:
; обращение к процедуре PP содержимое стека при входе в PP

Обращение к параметрам в процедуре можно осуществить с помощью регистра BP, присвоив ему значение SP. Но при этом мы испортим старое значение BP, которое может быть используется в основной программе. Поэтому следует вначале сохранить старое значение BP в стеке, а затем использовать его для доступа к параметрам, т.е тело процедуры должно начинаться следующими командами:



Для доступа к последнему параметру можно использовать выражение [BP + 4], например, mov AX, [BP = 4] ; ak AX. После "водных действий" в ПП идут команды, реализующие вспомогательный алгоритм, а за ними должны быть команды, реализующие "выходные действия":
---
pop BP; восстановить старое значение BP ret 2*k ; очистка стека от k параметров
PPendp; возврат в вызывающую программу
Необходимо помнить, что n в команде возврата ret n - это количество освобождаемых байтов в стеке, поэтому количество параметров должно быть умножено на длину параметра, т.е.

	программист сам должен подсчитать значение параметра в команде ret.
	Команда ret вначале считывает значение адреса возврата (av), а затем удаляет из стека параметры. Очистку стека можно выполнять не в ПП, а после выхода из нее, в основной программе, сразу после команды call PP, например, командой add SP*2.
	Каждый способ имеет свои достоинства и недостатки, если в ПП, то исполняемый код будет короче, если в основной программе, то можно вызвать ПП несколько раз с одними и теме же параметрами последовательными командами call.
	Для удобства использования параметров, переданных через стек, внутри ПП можно использовать директиву equ, чтобы при каждом обращении к параметрам не вычислять точное смещение относительно BP, например, так:
	Фрагмент программы:
	---------- 	
	push x
	push y
	push z
	call PP
	---------- 	
	Структура подпрограммы:
	PPproc near; процедура
	push BP
	mov BP, SP
	pp_x equ [BP + 8]
	pp_y equ [BP + 6]
	pp_z equ [BP + 4]
	---------- 	
	mov AX, pp_x; использование параметра x
	---------- 	
	pop BP
	ret 6
	ppendp
	Пример передачи параметров через стек.
	Пусть процедура заполняет нулями массив A[0..n-1] , основная программа обращается к ней для обнуления массивов X[0..99] и Y[0..49]. Через стек в ПП передается имя массива и его размер, размер можно передавать по значению, а имя массива нужно передавать по ссылке, так как этот параметр является и входным и выходным.
	; процедура zero_1
	zero_1proc
	push BP; входные
	mov BP, SP; действия
	push BX; сохранение значений
	push CX; регистров
	mov CX, [BP + 4]; CX = n считывание из стека
	mov BX, [BP + 6]; BX = A параметров
	m1:mov byte ptr [BX], 0; цикл обнуления
	inc BX; массива
	loop m1; A[0..n-1]

	; восстановление регистров и выходные действия
	pop CX
	pop BX
	pop BP
	Ret 4
	zero_1endp
	Фрагмент основной программы:
	X DB 100 dup (?)
	Y DB 50 dup (?)
	---------- 	
	lea AX, X; загрузка параметров:
	push AX; адреса массива X
	mov AX, 100; и его размера
	push AX; в стек
	call zero_1; обращение к ПП
	lea AX, Y; загрузка параметров для массива Y
	puah AX;
	mov AX, 50;
	push AX;
	call zero_1; обращение к ПП
	---------- 	
	О передаче параметров в ПП.
	1. Передача по значению:
	mov AX, word ptr value
	call PP
	2. Передача по ссылке:
	mov AX, offset value
	call PP
	3. Передача параметров по возвращаемому значению объединяет передачу по значению и по ссылке: процедуре передается адрес переменной, она делает локальную копию этого параметра, работает с этой копией, а в конце процедуры записывает эту копию по переданному адресу. Этот механизм оказывается эффективным, если процедуре приходится много раз обращаться к параметру в глобальной переменной.
	4. Передача параметров по результату заключается в том, что ПП передается адрес только для записи по этому адресу результата работы ПП.
	5. Передача параметров по имени макроопределения. Пример:
	name macro parametr
	mov AX, parametr
	nameendm
	Обращение к ПП может быть таким:
	name value; обращение к макро
	call PP; обращение к ПП
	6. Передача параметров отложенным вычислением, как и в случае передачи параметров по имени, процедура получает адрес ПП, вычисляющей значение параметра. Этот механизм чаще используется в системах искусственного интеллекта и в ОС.
	Использование локальных параметров.
	Если локальных параметров немного, то их размещают в регистрах, но если регистров недостаточно, то возможны

различные варианты: им можно отвести место в сегменте данных, но тогда большую часть времени эта область памяти не будет использоваться. Лучший способ - разместить локальные параметры в стеке на время работы ПП, а перед выходом из ПП их удалить. Для этого после входных действий в процедуре нужно уменьшить значение указателя на вершину стека SP на количество байтов, необходимых для хранения локальных величин и затем записывать их в стек и извлекать их оттуда можно с помощью выражений вида: [BP - n], где n определяет смещение локального параметра относительно значения BP.
Например, если предполагается, что ПП будет использовать 3 локальные параметра размером в слово, то стек графически можно представить так:

При выходе из процедуры перед выполнением завершающих действий нужно возвратить регистру SP его значение. Если в стеке хранятся и фактические и локальные параметры, то начало процедуры и ее завершение должно выглядеть следующим образом:

```
PPproc
push BP; сохранить старое значение BP
mov BP, SP; (SP) BP
sub SP, k1 ; отвести в стеке k1 байтов под
; локальные параметры
push AX; сохранить в стеке регистры,
----------; используемые в ПП
<тело процедуры>
pop AX; восстановить регистры
----------;
mov SP, BP; восстановить SP, т.е. освободить место в
; стеке от локальных параметров
pop BP ; восстановить BP равным до обращения к ПП
ret k2 ; очистка стека от фактических
; параметров и возврат в вызывающую программу
PPendp; конец ПП
```

Подсчет количества различных символов в заданной строке. Строка задана как массив символов. Начальный адрес ее передадим в ПП через регистр BX, длину строки через CX, а результат - через AX. Создадим процедуру, в которой выделяется 256 байтовый локальный массив L по количеству возможных символов. Каждому элементу этого массива будем присваивать единицу, если символ, цифровой код которого равен K, в заданной строке существует. Затем подсчитаем количество единиц в этом массиве. Вначале весь массив обнуляется. К первому элементу этого массива можно обратиться так: L_1 = [BP - 256] к K - му L_k = [BP - 256 + k]
Работая со строками, эту задачу можно решить проще.

	Count_s proc ;
	; входные действия
	push BP
	mov BP, SP
	sub SP, 256
	push BX
	push CX
	push SI
	; Обнуление локального массива
	mov AX, CX ; сохранение длины исходной строки
	mov CX, 256 ; возможное количество символов
	mov SI, 0; индекс элемента массива
	m1:mov byte ptr [BP - 256 + SI], 0 ;
	inc SI
	loop m1
	; просмотр заданной строки и запись 1 в локальный массив
	mov CX, AX ; длину строки в CX
	mov AX, 0
	m2:mov AL, [BX] ; код очередного символа в AL
	mov SI, AX ; пересылаем его в SI
	mov byte ptr [BP - 256 + SI], 1 ;пересылаем 1 в к -й элемент мас.
	inc BX
	loop m2;
	; подсчет количества 1 в локальном массиве
	mov AX, 0 ; результат будет в AX
	mov CX, 256 ; количество повторений цикла
	mov SI, 0 ; индекс массива в SI
	m3:cmp byte ptr [BP - 256 + SI], 1
	jne m4
	inc AX
	m4:inc SI
	loop m3
	; выходные действия
	pop SI; восстановление
	pop CX; регистров
	pop BX ;
	mov SP, BP ; освобождение стека от локальных параметров
	pop BP; восстановить старое BP
	ret
	const_s endp

Контрольный тест	Ответьте на вопросы: 1. Какой тип подпрограмм в Ассемблере? 2. Где можно размещать процедуры в программе? 3. Какие существуют способы передачи параметров? 4. Как можно передавать Параметры? 5. Охарактеризуйте передачу параметров через регистры 6. Охарактеризуйте передачу параметров через стек 7. Как используют локальные параметры?

Дата____10.11.2021 г _____ _____ _____

Подпись Ф.И.О. преподавателя