МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Институт прикладной математики и компьютерных наук
Кафедра «Прикладной математики и информатики»

Утверждено на заседании кафедры
«Информационная безопасность»
« 14 » января 2020 г., протокол № 6

Заведующий кафедрой
_____ В.И. Иванов

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению лабораторных работ
по дисциплине (модулю)
«*Архитектура компьютеров*»

основной профессиональной образовательной программы
высшего образования – программы бакалавриата

по направлению подготовки
01.03.02 Прикладная математика и информатика

с направленностью (профилем)
Прикладная математика и информатика

Форма обучения: очная

Идентификационный номер образовательной программы: 010302-01-20

Тула 2020 год

Разработчик методических указаний

Скобельцын С.А., доцент каф. ПМиИ, к.ф.-м.н. _____

(*ФИО, должность, ученая степень, ученое звание*) *(подпись*)

ЛАБОРАТОРНАЯ РАБОТА № 1

"Основы булевой алгебры. Системы счисления и операции с ними"

1. Цель и задачи работы

Освоить средства и приемы анализа элементов булевой алгебры, использования записи чисел в различных системах счисления и операции с ними.

2. Общие положения (теоретические сведения)

Типы данных
Фундаментальные типы данных
Фундаментальными типами для архитектуры IA-32 являются:

байт (byte)	8 бит
Слово (word)	16 бит
двойное слово (doubleword)	32 бит
четверное слово (quadword)	64 бит

При размещении данных в памяти используется прямой порядок байт (little-endian): младший байт располагается по наименьшему адресу. Адресом всего многобайтного числа является адрес его самого младшего байта. В приведенном примере по адресу 1234h располагается слово FFF0h, в котором младшему байту соответствует ячейка с адресом 1234h, а старшему байту - ячейка с адресом 1235h. Если данные по адресу 1234h интерпретировать как двойное слово, то ему соответствует число 008CFFF0h. Т.е. младший байт младшего слова располагается по адресу 1234h, старший байт младшего слова - по адресу 1235h и т.д.
Пример

7 0	
Содержимое	Адрес
AA	123Bh
5A	123Ah
3F	1239h
12	1238h
00	1237h
8C	1236h
FF	1235h
F0	1234h

При размещении данных в памяти рекомендуется использовать выравнивание, т.е. слова следует размещать по четным адресам, двойные слова - по адресам, кратным четырем, и т.п. Обращение к невыравненным данным вынуждает про-

цессор делать дополнительный цикл чтения или записи. Так, чтение двойного слова AA5A3F12h по адресу 1238h возможно за один цикл шины, тогда как чтение числа 3F12008Ch по адресу 1236h потребует как минимум двух циклов шины.

Языки высокого уровня обеспечивают выравнивание на уровне компилятора (для всей программы или для всего модуля). Ассемблер предоставляет директиву ALIGN, с помощью которой можно определять выравнивание различных участков данных.

Целые типы данных

Процессор, оперируя перечисленными фундаментальными типами данных, позволяет в программе интерпретировать эти значения как целочисленные. Целое число может считаться знаковым или беззнаковым. Большинство арифметических инструкций процессора работают одинаково для знаковых и для беззнаковых чисел, однако для некоторых инструкций (инструкции условных переходов, деление, умножение) существуют модификации для работы со знаковыми и беззнаковыми целыми числами.

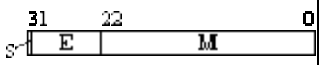
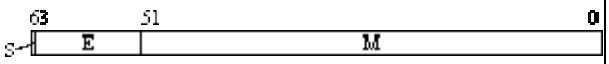
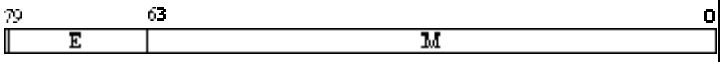
Фундаментальный тип кодирует беззнаковое число, если его значение интерпретируется как целое число от 0 до 2n-1, где n - разрядность типа. Для чисел со знаком используется комплементарное кодирование (дополнение до 1): т.е. отрицательному числу, модуль которого M, соответствует код NOT(M)+1, где NOT - операция побитового инвертирования. При этом по старшему разряду кода можно определить знак числа: 0 - неотрицательное число, 1 - отрицательное число. Знаковым числам соответствует диапазон -2n/2...+2n/2-1.

Фундаментальный тип	Целый тип	Диапазон
байт (byte)	символ со знаком (signed char)	-128...+127
	символ без знака (unsigned char)	0...255
Слово (word)	короткое со знаком (signed short)	-32768...+32767
	короткое без знака (unsigned short)	0...65535
двойное слово (doubleword)	целое со знаком (signed int)	-2147483648...+2147483647
	целое без знака (unsigned int)	0...4294967295

Вещественные типы данных

Встроенный математический сопроцессор (FPU) микропроцессоров IA-32 позволяет оперировать с вещественными типами данных. Вещественные числа кодируются в соответствии со стандартом IEEE-754. Старший разряд двоичного представления вещественного числа всегда кодирует знак числа. Остальная часть разбивается на две части: экспонента и мантисса. Вещественное число вычисляется как: $S*2^{E}*M$, где S - знаковый бит числа, E - порядок, M - мантис-

са. Обычно процессор оперирует с нормализованными вещественными числами, у которых 1<=M<2, так что целую часть мантиссы можно отбросить и кодировать только дробную часть (это делается в случае single precision и double precision). Экспонента кодируется со сдвигом на половину разрядной сетки, т.е. при 8-битной разрядности экспоненты код 0 соответствует числу -127, 1 - числу -126, ..., 255 - числу +126 (экспонента вычисляется как код-127).

тип	диапазон по модулю	двоичное представление
вещественное ординарной точности (single precision) - 32 бит	$1.18*10^{-38}$... $3.40*10^{38}$	
вещественное двойной точности (double precision) - 64 бит	$2.23*10^{-308}$... $1.79*10^{308}$	
вещественное расширенной точности (extended precision) - 80 бит	$3.37*10^{-4932}$ ... $1.18*10^{4392}$	

Указатели

Процессор оперирует с указателями двух типов: ближние (near) и дальние (far).

47 32	31 16	15 0	Тип
	Смещение		near pointer
селектор	Смещение		far pointer

Ближний указатель представляет собой 32-битное смещение (эффективный адрес) от начала сегмента. Ближние указатели используются в сплошной модели памяти, а также в сегментированной модели, если селектор сегмента задан неявно.

Дальний указатель (логический адрес) состоит из 16-битного селектора сегмента и 32-битного смещения (эффективного адреса). Дальние указатели используются в сегментированной модели памяти, когда требуется явное задание селектора сегмента.

В 16-битном (реальном) режиме адресации смещения имеют размер 16 бит, поэтому размер ближнего адреса составляет 16 бит, а дальнего - 32 бита.

Битовое поле - непрерывная последовательность битов, в которой каждый бит рассматривается как независимая переменная. Оно может начинаться с любого бита любого байта и может быть длиной до 32 бит.

Строка - непрерывная последовательность битов, байтов, слов или двойных слов. Битовая строка может начинаться с любой позиции любого байта и содержать до 232-1 битов. Строка байтов может содержать байты, слова или двойные слова и иметь размер до 232-1 байт.

Двоично-десятичные числа

Микропроцессор позволяет оперировать с числами в двоично-десятичной кодировке (binary-coded decimal - BCD). Двоично-десятичное число считается неупакованным, если одна десятичная цифра кодируется одним байтом. Двоично-десятичное число считается упакованным, если одна десятичная цифра кодируется четырьмя битами. Целочисленное устройство микропроцессора оперирует с 8-битовыми упакованными (0...99) или неупакованными (0...9) BCD-числами без знака. FPU поддерживает 80-битные упакованные BCD-числа со знаком. Такое число содержит 18 десятичных цифр в битах 0...71, а бит 79 - бит знака (биты 72...78 не используются). Этот тип позволяет работать с целыми числами в диапазоне $-10^{18}+1$ ... $+10^{18}-1$.

3. Объекты исследования, оборудование, материалы и наглядные пособия

Объект исследования – средства и приемы анализа элементов булевой алгебры, использования записи чисел в различных системах счисления и операции с ними.

В качестве оборудования используются персональные компьютеры учебных классов кафедры ПМиИ (ауд. 12-207, 12-209, 12-211).

В качестве операционной системы используется операционная система MS Windows XP SP2.

Средства анализа и вычислений – системная программа Калькулятор, электронная таблица MS Excel.

Средства ввода: клавиатура или текстовый файл.

Средства ввода: экран ПК или текстовый файл.

4. Задание на работу (рабочее задание)

Представить во всех внутренних форматах представления чисел процессора типа IA-32 следующие числа:

123
-87
23.176
-99.321

Получить шестнадцатеричные и двоичные представления чисел.

Для вещественных чисел показать ошибки представления десятичных чисел в ЭВМ.

5. Ход работы (порядок выполнения работы)

В среде операционной системы Windows XP с использованием макроассемблера Microsoft Macro Assembler Version 7 / 8 разработать, отладить и провести обработку контрольных примеров.

Для всех задач и используемых нетривиальных процедур/функций разработать и решить контрольные программы/примеры, результаты выполнения которых очевидны или легко проверяются.

Сохранить результаты работы (решение основных и контрольных задач) в текстовом файле или в документе Word.

Составить отчет о выполнении работы.

6. Содержание отчета

Отчет должен содержать:

1. Титульный лист;
2. Формулировку цели и задач работы;
3. Индивидуальное задание на работу;
4. Описание использованных программных и аппаратных средств для выполнения работы;
5. Описание хода работы с указанием этапов и пояснениями используемых решений (методов, программ, процедур, библиотек);
6. Список использованных источников.

7. Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.
2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.
3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.

ЛАБОРАТОРНАЯ РАБОТА № 2

"Аппаратное обеспечение ПК. Диагностика и устранение неисправностей"

1. Цель и задачи работы

Освоить средства и приемы идентификации аппаратного обеспечения ПК. диагностики и устранения неисправностей.

2. Общие положения (теоретические сведения)

Идеи фон Неймана в отношении организации ЭВМ

Основы учения об архитектуре вычислительных машин были заложены Джон фон Нейманом.

Основы учения об архитектуре ЭВМ заложил выдающийся американский математик Джон фон Нейман.

Первая ЭВМ "Эниак" была создана в США в 1946 г. В группу создателей входил фон Нейман, который и предложил основные принципы построения ЭВМ:

- переход к двоичной системе счисления для представления информации
- принцип хранимой программы: программу вычислений предлагалось помещать в запоминающем устройстве, что обеспечивало бы автоматический режим выполнения команд и, как следствие, увеличение быстродействия.

Заметим, что ранее все вычислительные машины хранили обрабатываемые числа в десятичном виде, а программы задавались путём установки перемычек на специальной коммутационной панели.)

Нейман первым предложил хранение программы в числовой форме – в виде набора нулей и единиц – причём в той же памяти, что и обрабатываемые ею числа.

В результате реализации идей фон Неймана была создана классическая архитектура ЭВМ.

Принципы фон Неймана

Использование двоичной системы счисления в вычислительных машинах. Преимущество перед десятичной системой счисления заключается в том, что устройства можно делать достаточно простыми, арифметические и логические операции в двоичной системе счисления также выполняются достаточно просто.

Программное управление ЭВМ. Работа ЭВМ контролируется программой, состоящей из набора команд. Команды выполняются последовательно друг за другом. Созданием машины с хранимой в памяти программой было положено начало тому, что мы сегодня называем программированием.

Память компьютера используется не только для хранения данных, но и программ. При этом и команды программы и данные кодируются в двоичной

системе счисления, т.е. их способ записи одинаков. Поэтому в определенных ситуациях над командами можно выполнять те же действия, что и над данными.

Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы. В любой момент можно обратиться к любой ячейке памяти по ее адресу. Этот принцип открыл возможность использовать переменные в программировании.

Возможность условного перехода в процессе выполнения программы. Не смотря на то, что команды выполняются последовательно, в программах можно реализовать возможность перехода к любому участку кода.

Таким образом, формально перечень принципов фон Неймана может быть приведен в следующем виде:

- Принцип использования двоичной системы счисления для представления данных и команд.
- Принцип программного управления.
- Программа состоит из набора команд, которые выполняются процессором друг за другом в определенной последовательности.
- Принцип однородности памяти.
- Как программы (команды), так и данные хранятся в одной и той же памяти (и кодируются в одной и той же системе счисления - чаще всего двоичной). Над командами можно выполнять такие же действия, как и над данными.
- Принцип адресуемости памяти.
- Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка.
- Принцип последовательного программного управления
- Все команды располагаются в памяти и выполняются последовательно, одна после завершения другой.
- Принцип условного перехода (сам принцип был сформулирован задолго до фон Неймана Адой Лавлейз и Чарльзом Бэббиджем, однако Нейман добавил его в общую архитектуру).

Компьютеры, построенные на этих принципах, относят к типу фон-неймановских

Архитектура компьютера фон Неймана

Компьютер состоит из нескольких основных устройств (арифметико-логическое устройство, управляющее устройство, память, внешняя память, устройства ввода и вывода).

Рис. 1. Архитектура компьютера фон Неймана

Арифметико-логическое устройство - выполняет логические и арифметические действия, необходимые для переработки информации, хранящейся в памяти.

Управляющее устройство - обеспечивает управление и контроль всех устройств компьютера (управляющие сигналы указаны пунктирными стрелками).

Данные, которые хранятся в запоминающем устройстве, представлены в двоичной форме.

Программа, которая задает работу компьютера, и данные хранятся в одном и том же запоминающем устройстве.

Для ввода и вывода информации используются устройства ввода и вывода.

3. Объекты исследования, оборудование, материалы и наглядные пособия

Объект исследования – средства и приемы идентификации аппаратного обеспечения ПК. диагностики и устранения неисправностей.

В качестве оборудования используются персональные компьютеры учебных классов кафедры ПМиИ (ауд. 12-207, 12-209, 12-211).

В качестве операционной системы используется операционная система MS Windows XP SP2.

Среда разработки – макроассемблер "Microsoft Macro Assembler Version 7 / 8".

Средства ввода: клавиатура или текстовый файл.

Средства ввода: экран ПК или текстовый файл.

4. Задание на работу (рабочее задание)

Познакомиться с аппаратной частью системного блока ПК. Освоить приемы разборки/сборки устройств системного блока. Научиться диагностировать неисправности компьютера по сигналам динамика.

5. Ход работы (порядок выполнения работы)

1) подключить к системному блоку СБ0 кабель питания, монитор (разъем монитора необязательно закреплять винтами!) и клавиатуру от ПК 207-10
2) включить ПК СБ0, убедится, что на нем загружается ОС Windows 98
3) выключить
4) повторно включить и клавишей Pause приостановить процесс загрузки на первом экране работы программы самотестирования BIOS, где видна информация о системной плате, BIOS, процессоре, объеме оперативной памяти, автоопределяеммых IDE-устройствах; проанализировать информацию;
5) нажать любую клавишу и повторно нажать клавишу Pause для остановки программы самотестирования оборудования на 2-м экране, где выдается информация о процессоре, объеме оперативной памяти (распределение между басовой и расширенной частью), кеше, установленных устройствах IDE (в частности об объеме жесткого диска), установленных PCI-устройствах и закреплении за ними прерываний (IRQ); проанализировать информацию
6) выключить ПК
7) отсоединить все кабели от системного блока
8) снять кожух системного блока СБ0
9) положить системный блок на бок
10) продемонстрировать все компоненты, названные в разделе "основные понятия" выше
11) отключить кабель питания НЖМД
12) подключить к системному блоку кабель монитора и кабель питания
13) нажать кнопку включения
14) обратить внимание на короткий сигнал динамика, обозначающий начало нормального процесса загрузки
15) посмотреть на содержимое вывода программ самотестирования и начальной загрузки на мониторе
16) выключить ПК кнопкой включения
17) отключить кабель питания системного блока
18) отведя в стороны белые фиксаторы модуля памяти (DIMM SDRAM), извлечь его, удерживая за торцы печатной платы (по возможности, не касаясь микросхем и электрических контактов на модуле); обратить внимание на расположение микросхем на плате модуля по отношению к процессору (они расположены с противоположной стороны)
19) подключить кабель питания системного блока

20) включить ПК кнопкой включения
21) убедится, что динамик периодически издает относительно продолжительные звуки, указывая на проблемы с RAM; на мониторе не появляется никакого изображения
22) выключить ПК кнопкой включения удерживая ее до остановки вентиляторов БП и на процессоре
23) отключить кабель питания системного блока
24) установить модуль памяти на место в следующем порядке: вначале белые фиксаторы модуля должны быть разведены; ориентируя модуль микросхемами от процессора, удерживая его за торцы печатной платы установить DIMM в направляющие 2-го разъема памяти (направляющие расположены по краям разъема); затем втолкнуть модуль в разъем окончательно, поочередно надавливая ("раскачивая") большими пальцами рук на верхнюю часть модуля у краев разъема; при этом белые фиксаторы модуля памяти должны подняться (усилия не должны быть слишком велики с тем, чтобы механически не повредить модуль, разъем или системную плату); в конце поджать фиксаторы с обеих сторон в направление модуля
25) включить кабель питания системного блока
26) включить ПК кнопкой включения
27) убедится, что динамик издает только один короткий звук и процесс загрузки начинается (есть изображение на мониторе)
28) выключить ПК кнопкой включения, перейти к п.30
29) если в п.27 загрузка не происходит и динамик издает продолжительные звуки, указывая на проблемы с RAM, то следует повторить пункты 22-28 до получения нормального процесса загрузки (вначале попробовать установить модуль памяти плотнее, надавливая на него дополнительно; потом попробовать снять и установить снова; в конце концов можно установить модуль в другой разъем)
30) отключить кабель питания системного блока
31) снять видеоадаптер: отключить монитор от системного блока; отвернуть винт крепления скобы видеокарты к задней стенке корпуса; вытягивая за внутренний конец видеокарты (обращенный к HDD) и выталкивая ее вверх за VGA-разъем со стороны задней стенки корпуса извлечь видеоадаптер из разъема AGP
32) включить кабель питания системного блока
33) включить ПК кнопкой включения
34) убедится, что динамик издаст сигнал, состоящий из нескольких коротких звуков (на некоторых системных платах такой сигнал периодически повторяется), что показывает, что программа тестирования оборудования обнаружила отсутствие (неисправность) видеоадаптера
35) выключить ПК кнопкой включения удерживая ее до остановки вентиляторов БП и на процессоре
36) отключить кабель питания системного блока
37) установить видеоадаптер обратно в системный корпус, закрепить винтом металлическую скобу видеокарты к задней стенке системного блока

38) подключить монитор
39) включить ПК кнопкой включения
40) убедится, что динамик издает только один короткий звук и процесс загрузки начинается (есть изображение на мониторе); прейти к п. 42
41) если процесс загрузки не начинается, то пп. 36-40 повторить
42) (необязательно) поступая аналогично пп. 30-40, снять и установить модуль процессора вместе с блоком адаптера из (в) разъема Slot 1; убедится, что при снятом (неисправном) процессоре или нарушенных контактов в его разъеме система становится полностью неработоспособной и при попытке включения динамик не издает никаких сигналов, т.к. не может работать программа тестирования BIOS
43) (необязательно) операции пп. 16-42 повторяет учащийся (учащиеся)
44) отключить все кабели от системного блока
45) подключить кабель питания НЖМД
46) поставить системный блок вертикально
47) установить кожух
48) подключить к системному блоку кабель питания, монитор и клавиатуру
49) включить ПК, убедится, что ОС Windows 98 загружается
50) выключить ПК
51) отключить все кабели от системного блока СБ0
52) подключить кабель питания, монитор и клавиатуру к системному блоку ПК 207-10

6. Содержание отчета

Отчет должен содержать:

1. Титульный лист;
2. Формулировку цели и задач работы;
3. Индивидуальное задание на работу;
4. Описание использованных программных и аппаратных средств для выполнения работы;
5. Описание хода работы с указанием этапов и пояснениями используемых решений (методов, программ, процедур, библиотек);
6. Список использованных источников.

7. Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.
2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.

3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.

ЛАБОРАТОРНАЯ РАБОТА № 3

"Этапы и среда разработки программ в MASM для ОС Windows"

1. Цель и задачи работы

Изучить этапы и среда разработки программ в MASM для ОС Windows

2. Общие положения (теоретические сведения)

В соответствии с типовой процедурой подготовки программы:

1. разработка алгоритма работы программы, обеспечивающего автоматизацию решения некоторой задачи (прикладной – инженерной, экономической, научной – или системной);
2. написание текста программы на некотором языке программирования;
3. перевод текста программы на машинный носитель (формирование исходного модуля программы);
4. компиляция программы (получение объектного модуля программы) – перевод ее на язык команд процессора, на котором предполагается ее эксплуатация;
5. компоновка программы (формирование выполняемой программы) – сборка программы из отдельных объектных модулей: библиотечных и полученных из модулей исходной программы с учетом специфики исполняющей среды (аппаратной платформы, операционной системы, режима исполнения);
6. отладка программы – выполнение программы для некоторых наборов исходных данных, позволяющее убедиться в правильной работе программы хотя бы для ряда контрольных наборов исходных данных (контроль наличия и исключение логических ошибок) (обычно этот этап предполагает неоднократное выполнение шагов 2-6);

для реализации этапов 2-3 предлагается использовать встроенный текстовый редактор файлового менеджера FAR, для реализации этапа 4 – компилятор Macro assembler (ml.exe) [4] интегрированной системы разработки Microsoft Visual Studio 2005 (можно использовать версии 2003, 2008, 2010), а для реализации этапа 5 – компоновщик системы Visual Studio – link.exe и её систему библиотек. Заметим, что визуальная среда разработки Visual Studio является всего лишь удобной оконной надстройкой над отдельными компонентами разработки системы MS Visual Studio, многие из которых можно использовать и независимо.

Редактирование текста программы (шаги 2, 3)

Как для большинства средств разработки (C, C++, Basic, Pascal, Delphi, Java, C#, …, но не VBA, Power Builder, хранимых процедур SQL,…, в которых тексты программ интегрированы в отдельные объекты проектов, хранимые в специальных форматах) исходный текст программы на языке ассемблера должен быть подготовлен в виде текстового файла. Обычно для файлов исходных текстов программ на ассемблере используют расширение .asm, например, first_program.asm, test_1.asm, hello.asm.

Для подготовки исходных текстов программ на языке ассемблера в среде Windows можно использовать любой текстовый редактор: notepad, WordPad, Word, редактор системы разработки Visual Studio. Однако, поскольку предполагается разрабатывать в основном консольные программы, то рекомендуется использовать

встроенный текстовый редактор файлового менеджера FAR (Files and ARchives manager). Это обусловлено следующими причинами:

FAR-менеджер не требует установки и для обеспечения его полной функциональности в качестве консольного окна и редактора текста достаточно скопировать каталог FAR-менеджера (обычно \Program Files\Far) с одного ПК на другой (для того, чтобы на новом ПК сохранить и настройки со старого, необходимо в каталоге ранее установленного экземпляра выполнить командный файл SaveSettings.bat, а в каталоге скопированного экземпляра выполнить командный файл RestoreSettings.bat);

редактор FAR обладает достаточно широкими средствами редактирования текста, не перегружая разработчика излишними средствами форматирования текста;

в редактируемом тексте можно легко менять кодировку символов с OEM-DOS на Win-1251 и наоборот;

среда программы FAR представляет собой естественное консольное окно для выполнения компонент системы разработки и консольной программы – результата разработки;

FAR позволяет в рамках одного экземпляра программы использовать одновременно несколько окон редактирования и основное окно – консоль (переключение – Ctrl+Tab);

FAR предлагает ряд удобных средств для манипуляции командами командной строки (копирование и вставка с использованием буфера обмена; редактирование команды в командной строке; вставка в командную строку текущего имени файла с панели; вставка в командную строку полного имени каталога левой или правой панели; воспроизведение ранее набранных команд последовательным перебором /Up, Down/ или выбором из меню /Alt+F8/).

Компиляция программы (этап 4)

Компиляцию программ на ассемблере будем выполнять компилятор Macro assembler.

Например, простейшей командой компиляции программы prog.asm является команда в консольном окне ОС (окне программы cmd или FAR)

```
>ml prog.asm
```

Предполагается, что в системе установлена какая-либо версия системы разработки MS Visual Studio.

В результате компиляции исходного текста программы формируется объектный файл (модуль) prog.obj.

Заметим, что если объектный файл не формируется, то значит – компилятором обнаружены синтаксические ошибки в написании программы. Диагностические сообщения об ошибках выводятся в консоль. После анализа сообщений может понадобиться исправление исходного текста программы (файла prog.asm) и повторная компиляция.

Компоновка программы (этап 5)

Формирование выполняемой программы (файла с расширением exe) проводится компоновщиком link.exe из комплекта той же системы MS Visual Studio.

Простейшей командой компоновки программы prog является команда

```
>link prog.obj
```

В результате компоновки объектного модуля программы формируется выполняемый файл prog.exe.

Если выполняемый файл не формируется, то говорят об ошибках этапа компоновки. По анализу диагностических сообщений, выводимых на консоль, может понадобиться указать какие-либо дополнительные параметры программы link.exe или изменить исходный текст программы и повторно перекомпилировать ее.

Отладка программы проводится выполнением prog.exe и тестированием полученных результатов. Если необходимо, проводится редактирование исходного текста, компиляция, компоновка и выполнение повторно.

3. Объекты исследования, оборудование, материалы и наглядные пособия

Объект исследования – этапы и среда разработки программ в MASM для ОС Windows.

В качестве оборудования используются персональные компьютеры учебных классов кафедры ПМиИ (ауд. 12-207, 12-209, 12-211).

В качестве операционной системы используется операционная система MS Windows XP SP2.

Среда разработки – макроассемблер "Microsoft Macro Assembler Version 7 / 8".

Средства ввода: клавиатура или текстовый файл.

Средства ввода: экран ПК или текстовый файл.

4. Задание на работу (рабочее задание)

1-й шаг: создание текста программы

Включить программу FAR. Закрыть левую панель программы (Ctrl+F1). Перейти в корневой каталог диска C:\ (>CD C:\ или Alt+F2 C). Здесь и далее в форме ">команда" указывается команда "команда" в командной строке FAR. При этом на открытой (правой) панели FAR'а должно отображаться содержимое корневого каталога диска C:.

Создать в корневом каталоге диска C: каталог группы C:\520111 (F7, 520111, [Enter]). Сделать текущим каталог C:\520111. В каталоге группы C:\520111 создать каталог студента (F7, Петров, [Enter]). Сделать текущим каталог студента (выделить в панели имя каталога Петров, [Enter] или >cd C:\520111\Петров). При этом приглашение командной строки FAR должно иметь вид "C:\520111\Петров>".

Создание текста программы. Создать новый файл (Shift+F4, prg_01.asm, [Enter]). Здесь prg_01.asm – название файла с исходным текстом программы на ассемблере. В открывшемся окне редактора набрать текст программы:

```
.686
.MODEL        FLAT, C
INCLUDE       stdio.inc

.STACK
```

```
          .DATA
Text1     BYTE          "Hello, World!",0

          .CODE

main      PROC

          INVOKE        puts, ADDR Text1
          ret

main      ENDP

          END ; program
```

Завершить редактирование: сохранить файл (Shift+F2), закрыть редактор (F10).

Собственно, реально выполняемыми командами процессора в представленной программе являются команды INVOKE и ret. При этом INVOKE обеспечивает вызов функции puts библиотеки функций языка программирования C (C Run-Time Library, CRT Library). С MS Visual Studio 2005 CRT Library поставляется в виде динамической библиотеки msvcr80.dll, а с MS Visual Studio 2003 – msvcr71.dll, Visual Studio 2010 – msvcr100.dll. Библиотека не является стандартной системной библиотекой, но используется большим числом приложений в Windows XP. Поэтому практически всегда в системе установлена CRT Library (даже тогда, когда не установлена система разработки MS Visual Studio).

Заметим, что команда INVOKE не является командой процессора. Она является директивой, которая формирует несколько команд процессора. Эти команды можно увидеть в листинге программы (листинг – необязательный файл с расширением lst, формируемый при компиляции). Макрокоманда INVOKE в данной программе реализуется следующей последовательностью команд процессора (см. файл prg_01.lst):

```
          push          OFFSET Text1
          call          puts
          add           esp,4
```

Здесь команда push помещает в стек смещение (адрес) первого байта строки Text1, командой call передается управление стандартной функции puts. Она выводит в консоль как символы содержимое байтов последовательно расположенных в ОП, начиная с байта с адресом, установленным командой push, до тех пор, пока не встретится байт со значением 0 (спецификация функции puts приводится в документации по языку C, а также в справочной системе Microsoft Developer Network – MSDN; в Интернет см. [5]).

Второй выполняемой командой программы (точнее четвертой) является команда процессора ret, которая возвращает управление из основной процедуры программы пользователя (main) управление в вызывающий модуль _mainCRTStartup.

Заметим, среду выполнения консольной программы в системе Windows организует объектный модуль _mainCRTStartup, который включается в программу (exe-файл) на этапе компоновки из библиотеки msvcrt.lib. При запуске программы, после загрузки ее в ОП управление получает модуль _mainCRTStartup. Он выполняет необходимые операции по формированию среды выполнения консольной программы и передает управление основной процедуре программы пользователя – main. Процедура main после выполнения возвращает управление модулю-менеджеру _mainCRTStartup (командой ret), который завершает работу exe-программы вызовом

соответствующих функций ОС.

Команда

```
Text1     BYTE          "Hello, World!",0
```

является директивой, которая резервирует область ОП в 14 байт. В первые 13 байтов области будут помещены символы "Hello, World!", а в последний байт будет записано нулевое значение, обозначающее конец строки в соответствии с правилами организации строк в языке программирования С. Для ссылки на 1-й байт этой области вводится имя Text1.

Остальные команды программы

```
          .686
          .MODEL        FLAT, C
          INCLUDE       stdio.inc
          .STACK
          .DATA
          .CODE
main      PROC
main      ENDP
          END ; program
```

являются директивами, которые описывают особенности организации программы: структуру и типы компонент.

Частью программы при этом становится и файл stdio.inc, который должен находится в подкаталоге INC текущего каталога. Файл stdio.inc содержит спецификации (прототипы) функций ввода-вывода библиотеки CRT Library. В частности в нем описана функция puts.

Файл stdio.inc готовится заранее и записывается в подкаталог INC до начала компиляции.

2-й шаг: создание командного файла компиляции/компоновки программы

Для компиляции программы будем использовать командный (.bat) файл, который позволяет одной командой выполнить несколько программ/команд с указанием необходимых параметров.

Создадим командный файл ca.bat так, чтобы в результате выполнения команды

```
>ca prg_01.asm[Enter]
```

вначале формировался файл объектного кода программы prg_01.obj, а затем – и выполняемый файл prg_01.exe.

В объектный код программа переводится компилятором – программой ml.exe. Из файла объектного кода выполняемый exe-файл формируется компоновщиком – программой link.exe.

Поэтому в командном файле ca.bat нежно последовательно вызвать программы ml.exe и link.exe с необходимыми параметрами.

Не делая никаких предположений о составе переменных окружения перед выполнением программ ml.exe и link.exe необходимо обеспечить обнаружение самих программ и необходимых для них библиотек операционной системой с помощью установки переменных окружения.

Порядок создания файла ca.bat. Создать новый файл (Shift+F4, ca.bat, [Enter]). В открывшемся окне редактора набрать текст командного файла:

```
@SET INCLUDE=.\INC
```

```
@SET PATH=C:\Program Files\Microsoft Visual Studio
8\VC\bin;C:\Program *Files\Microsoft Visual Studio 8\Common7\IDE
@SET LIB=C:\Program Files\Microsoft Visual Studio
*8\VC\PlatformSDK\Lib\;C:\Program Files\Microsoft Visual Studio
*8\VC\lib\;
@ml.exe /c /nologo /Sa /Fl %1
@link.exe /NOLOGO /SUBSYSTEM:CONSOLE /MACHINE:X86 %~n1.obj
msvcrt.lib *kernel32.lib user32.lib gdi32.lib shell32.lib
@mt.exe /nologo -manifest %~n1.exe.manifest -
*outputresource:%~n1.exe;#1 >nul 2>nul
```

Здесь знак * в первой позиции строки обозначает продолжение предыдущей строки и не должен вноситься в сам текст командного файла. После набора текста командного файла следует завершить редактирование: сохранить файл (Shift+F2), закрыть редактор (F10).

Первые 3 команды полученного batch-файла:

```
@SET INCLUDE
@SET PATH
@SET LIB
```

формируют переменные окружения на время выполнения командного файла.

Заметим, что пути, указываемые в переменных PATH, LIB зависят от версии MS Visual Studio и каталога установки. Здесь указан вариант для версии Visual Studio 2005 когда системным диском является C:, а установка Visual Studio выполнена в каталог по умолчанию (C:\Program Files\Microsoft Visual Studio 8).

Например, в случае, если на компьютере используется Visual Studio 2003, установленная в корневой каталог диска D:, то команды SET PATH и SET LIB должны иметь вид:

```
@SET PATH=D:\Microsoft Visual Studio .NET 2003\Vc7\bin\;
*D:\Microsoft Visual Studio .NET 2003\Common7\IDE
@SET LIB=D:\Microsoft Visual Studio .NET 2003\Vc7\PlatformSDK\Lib;
*D:\Microsoft Visual Studio .NET 2003\Vc7\lib
```

Переменная INCLUDE указывает каталог, в котором будет выполняться поиск файлов типа .inc, включаемых в исходный текст директивами INCLUDE. Переменная PATH определяет каталоги, в которых размещаются используемые выполняемые файлы (ml.exe, link.exe, mt.exe) и необходимые им несистемные dll-библиотеки. Переменная LIB определяет каталоги, в которых будет выполняться поиск библиотек объектных модулей (файлов типа .lib) компоновщиком link.exe при разрешении внешних ссылок формируемой exe-программы.

Компиляция выполняется командой

```
@ml.exe /c /nologo /Sa /Fl %1
```

Здесь ml.exe – имя программы-компилятора. Параметр /c указывает, что от ml.exe требуется только компиляция программы (без /c компилятор автоматически вызывает компоновщик). Параметр /nologo блокирует вывод на экран информации о компиляторе. Параметр /Fl указывает на необходимость формирования листинга программы (необязательного файла с расширением lst, используемого для анализа получаемого кода программы и для отладки). Параметр /Sa указывает на необходимость формирования листинга в максимально подробной форме (в частности, для того, чтобы были видны команды процессора, генерируемые директивами и макрокомандами). Параметр %1 является ссылкой на 1-й параметр самого командного файла. Предполагается, что 1-й параметр – имя исходного файла программы (например, prg_01.asm).

Компоновка выполняется командой

```
@link.exe /NOLOGO /SUBSYSTEM:CONSOLE /MACHINE:X86 %~n1.obj
msvcrt.lib *kernel32.lib user32.lib gdi32.lib shell32.lib
```

Здесь link.exe – имя программы-компоновщика. Параметр /NOLOGO аналогичен соответствующему параметру у ml.exe. Параметр /SUBSYSTEM:CONSOLE указывает тип формируемого exe-файла (в данном случае – консольное приложение). Параметр %~n1.obj формирует имя объектного файла (при этом %~n1 – часть имени файла, указанного в 1-м параметре командного файла без расширения; .obj – расширение объектного файла). Имена msvcrt.lib, kernel32.lib, … представляют собой имена файлов-библиотек, которые используются для разрешения внешних ссылок. Сами файлы должны находится в каталогах, указанных в переменной окружения LIB.

Последняя команда файла ca.bat

```
@mt.exe /nologo -manifest %~n1.exe.manifest -
*outputresource:%~n1.exe;#1 >nul 2>nul
```

предназначена для включения так называемого манифеста в сформированный на этапе компоновки exe-файл в качестве ресурса (для Visual Studio начиная с 2005).

Файл манифеста (с расширением .manifest; например, для программы prg_01.exe – prg_01.exe.manifest) формируется для exe-файла компоновщиком link.exe. Он позволяет контролировать версии используемых dll-библиотек. Без манифеста, программа, использующая динамические библиотеки, и подготовленная в Visual Studio, начиная с версии 2005, не может быть выполнена. Указанная команда (программа mt.exe – manifest tool – менеджер манифеста) позволяет включить сведения из манифеста в тело exe-файла. После такой модификации для выполнения программы отдельный файл манифеста не требуется.

Для Visual Studio 2003 последнюю команду не надо добавлять к файлу ca.bat.

3-й шаг: компиляция/компоновка программы

Перед началом компиляции необходимо в каталоге студента сделать каталог INC и записать в него файл stdio.inc из приложения к данному методическому указанию.

Компиляция и компоновка программы, подготовленной на 1-м шаге выполняется командой:

```
C:\520111\Петров>ca.bat prg_01.asm
```

Подчеркнем, команда должна выполняться в каталоге, где расположены файлы исходной программы prg_01.asm, командный файл ca.bat и каталог INC.

При компиляции программы формируются файлы:

prg_01.obj – объектный файл (код) программы, содержащий код команд процессора и информацию необходимую для компоновки;

prg_01.lst – листинг программы, в котором можно анализировать результаты компиляции (ошибки, полученный объектный код).

Полезно проанализировать листинг программы. Основной интерес обычно представляет первая часть листинга, в которой представлен перевод данных и текста программы в кодовое представление данных и команд процессора в ОП.

Для программы prg_01.asm эта часть имеет вид

```
.686
```

```
                                .MODEL      FLAT, C
                                INCLUDE     stdio.inc
      ...
                        C
                        C puts  PROTO       C :PTR SBYTE
                        C
      ...
                                .STACK

0000                            .DATA
0000  48 65 6C 6C 6F  Text1     BYTE        "Hello, World!",0
      2C 20 57 6F 72
      6C 64 21 00

0000                            .CODE

0000                    main    PROC

                                INVOKE      puts, ADDR Text1
0000  68 00000000 R             * push      OFFSET Text1
0005  E8 00000000 E             * call      puts
000A  83 C4 04                  * add       esp, 04h
000D  C3                        ret

000E                    main    ENDP

                                END ;program
```

Из листинга видно, что в соответствии с директивой INCLUDE stdio.inc, компилятор определил прототип вызова функции (процедуры) puts. Он выглядит так:

```
puts      PROTO          C :PTR SBYTE
```

В разделе .DATA можно увидеть относительное смещение переменной Text1 в сегменте данных (0000) и порядок последовательного размещения символов строки Text1 в ОП. Легко видеть, что шестнадцатеричный код символа "H" = 48, символа "!" – 21.

Команды, порождаемые директивой INVOKE, помечены в листинге символом "*". Видно, как ранее было сказано, что это 3 команды: push, call и add.

При этом шестнадцатеричный код команды push – 68 (он занимает один байт). Код команды add с регистром esp – двухбайтовый – 83C4. Код команды ret – C3.

Заметим, в части адреса команд push, call стоят нулевые значения (00000000). Но в первом случае это означает, что смещение переменной Text1 в сегменте данных равно 0 (что отмечено выше), а во втором – для команды call – это означает, что место размещения процедуры puts в ОП в момент выполнения программы компилятору не известно.

Результатом выполнения компоновки являются также два файла:

prg_01.exe.manifest – файл манифеста для контроля версий динамических библиотек;

prg_01.exe – выполняемый файл программы.

Файл манифеста – xml-файл в котором находится спецификация необходимой

для выполнения программы prg_01.exe динамической библиотеки CRT Library:

```
<assemblyIdentity type='win32' name='Microsoft.VC80.CRT'
version='8.0.50608.0' processorArchitecture='x86'
publicKeyToken='1fc8b3b9a1e18e3b' />
```

После интеграции манифеста в программу (выполнения утилиты mt.exe) файл prg_01.exe оказывается относительно независимым. Поэтому файл манифеста prg_01.exe.manifest можно удалить.

Задание на самостоятельную работу

1) Проанализировав прототип функции printf из той же библиотеки CRT Library (в файле stdio.inc), изменить программу prg_01.asm (сделав копию и назвав prg_02.asm, prg_03.asm) так, чтобы вывод выполнялся в соответствии с вызовом функции printf в программе на C:

а) printf("Hello,\nWorld!");

(символ \n – LF – имеет код 0Ah)

б) printf("Hello, %s!",s);

где s – переменная в сегменте данных – строка байт "World",0

2) В примере prg_04.asm программа выводит в окно сообщений Windows (MessageBox) текст, который указывается в качестве 1-го параметра командной строки при запуске программы. Например, при запуске программы командой

```
C:\520111\Петров>prg_04.exe assembler[Enter]
```

будет выведено окно сообщений с заголовком "Message" и текстом в окне "assembler".

Необходимо изменить программы prg_01.asm –> prg_05.asm и prg_03.asm –> prg_06.asm так, чтобы при вызове:

```
C:\520111\Петров>prg_05.exe assembler[Enter]
C:\520111\Петров>prg_06.exe assembler[Enter]
```

они выводили в консоль соответственно:

assembler

Hello, assembler!

Т.е. весь текст в 1-м случае и часть текста во втором – должны браться из командной строки.

5. Ход работы (порядок выполнения работы)

В среде операционной системы Windows XP с использованием макроассемблера Microsoft Macro Assembler Version 7 / 8 разработать, отладить и провести обработку контрольных примеров.

Для всех задач и используемых нетривиальных процедур/функций разработать и решить контрольные программы/примеры, результаты выполнения которых очевидны или легко проверяются.

Сохранить результаты работы (решение основных и контрольных задач) в текстовом файле или в документе Word.

Составить отчет о выполнении работы.

6. Содержание отчета

Отчет должен содержать:

1. Титульный лист;
2. Формулировку цели и задач работы;
3. Индивидуальное задание на работу;
4. Описание использованных программных и аппаратных средств для выполнения работы;
5. Описание хода работы с указанием этапов и пояснениями используемых решений (методов, программ, процедур, библиотек);
6. Список использованных источников.

7. Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.
2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.
3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.

ЛАБОРАТОРНАЯ РАБОТА № 4

"Разработка программы на языке ассемблера с использованием системных функций ввода-вывода"

1. Цель и задачи работы

Освоить средства и приемы разработки программ на языке ассемблера с использованием системных функций ввода-вывода.

2. Общие положения (теоретические сведения)

Пример использования системных функций ввода-вывода.

```
                .686
                .MODEL          FLAT,C
                INCLUDE         in_out.inc

                .STACK

                .DATA
NN              =               200
LF              =               0Ah
Message         BYTE            "Введен текст:[%s]",0

                .DATA?
stdin           DWORD           ?
Text2           BYTE            NN DUP (?)

                .CODE

main            PROC

;get stdin
                INVOKE          __iob_func              ;get stdin
                mov             stdin,eax
;input
                mov             ebx,OFFSET Text2;start of Text2
;repeating while not LF
loop1:
                INVOKE          fgetc,stdin             ;get character from stdin
                cmp             al,LF
                je              continue                ;character is LF !
                mov             [ebx],al                ;moving character to RAM (to Text2)
                inc             ebx                                     ;next free byte
                jmp             loop1                           ;to next character input
```

```
continue:
        mov         BYTE PTR [ebx],0
;output
        INVOKE      printf,ADDR Message,ADDR Text2
        ret

main    ENDP

        END ;program
```

3. Объекты исследования, оборудование, материалы и наглядные пособия

Объект исследования – приемы разработки программ на языке ассемблера с использованием системных функций ввода-вывода.

В качестве оборудования используются персональные компьютеры учебных классов кафедры ПМиИ (ауд. 12-207, 12-209, 12-211).

В качестве операционной системы используется операционная система MS Windows XP SP2.

Среда разработки – макроассемблер "Microsoft Macro Assembler Version 7 / 8".

Средства ввода: клавиатура или текстовый файл.

Средства ввода: экран ПК или текстовый файл.

4. Задание на работу (рабочее задание)

В соответствии с индивидуальным заданием написать программу, которая

1) выводит на консоль (stdout) запрос вида "Введите строку:" (в примерах - Prompt);

2) вводит в отведенную область сегмента данных программы (в примерах - Text2)

строку символов (<=100 символов), вводимую пользователем с консоли (stdin)

(ввод завершается нажатием [Enter]);

3) выводит на консоль строку, содержащую фиксированный текст (например, "Введена строка:")

(в примере Message) и текст, введенный на шаге 2) (в примерах - Text2).

Т.е. схематично диалог с программой должен выглядеть так:

>prog.exe[Enter]

PromptText2[Enter]

MessageText2

Примечания:

1)

Некоторые функции при выводе переводят каретку. Поэтому допустим и вариант:

>prog.exe[Enter]
Prompt
Text2[Enter]
MessageText2

2)
В том случае, когда для вывода используется функция MessageBox текст Prompt (или MessageText2) должен выводится в окно MessageBox (см. пример puts_ReadFile_MessageBox.asm)

3)
Вывод строки MessageText2 обязательно выполнять одним вызовом функции
(в одном цикле, если функция выводит посимвольно).

4)
Пример выполнения программной части работы представлен в программах
puts_gets_puts.asm и puts_ReadFile_MessageBox.asm из приложения.

Индивидуальное задание определяет 3 функции, которые должны быть использованы
для выполнения шагов 1), 2), 3).

Варианты			
	1)	2)	3)
1	WriteFile	_cgets_s	putc
2	WriteConsole	_cscanf_s	putchar
3	MessageBox	gets	_fputchar
4	putc	fread	WriteFile
5	fputc	_cgets_s	WriteConsole
6	_putch	_cscanf	MessageBox
7	putchar	ReadFile	fprintf
8	_fputchar	ReadConsole	puts
9	_cputs	ReadFile	fputc
10	_cprintf	ReadConsole	putc
11	_cprintf_s	ReadFile	_putch
12	puts	ReadConsole	putchar
13	fputs	ReadFile	putc
14	_write	ReadConsole	_fputchar
15	fwrite	ReadFile	putc
16	printf	ReadConsole	fputc

17	printf_s	ReadFile	_putch	
18	fprintf	ReadConsole	putchar	
19	fprintf_s	ReadFile	_fputchar	
20	WriteFile	gets	putchar	
21	WriteConsole	gets_s	_fputchar	
22	MessageBox	fgets	putc	
23	putc	fgets	WriteFile	
24	fputc	_read	WriteConsole	
25	_putch	fread	MessageBox	
26	putchar	ReadFile	fprintf	
27	_fputchar	ReadConsole	printf_s	
28	_cputs		ReadFile	putchar
29	_cprintf	ReadConsole	putchar	
30	_cprintf_s	ReadFile	putc	

Указания к выполнению работы

При выполнении работы с использованием компилятора и библиотек MS Visual Studio 2003

допустима замена функций с суфиксом "_s" на аналогичные без суффикса.

Помимо собственно функций ввода-вывода в ряде вариантов требуется использование функций

__iob_func - для определения указателей на структуры FILE для стандартных файлов ввода-вывода

(stdin, stdout, stderr)

GetStdHandle - для определения Windows-дескрипторов стандартных файлов ввода-вывода

(stdin, stdout, stderr)

_fileno - для определения C-дескриптора файла по указателю на структуру FILE

strlen - для определения длины C-строки

Обратите внимание, если функция возвращает значение, то оно оказывается в регистре eax.

Информация о функциях, используемых в работе, приведена в рабочей книге Excel Lab_04.xls

Примеры использования некоторых функций представлены в приложении.

Для обращения ко всем функциям в примерах используется файл INC\in_out.inc,

который содержит описание необходимых констант, типов и структуры FILE (_iobuf).

Для своей программы сформировать in_out.inc с минимально необходимым количеством определений.

Особенности команд, использованных в программах-примерах, см. в Руководстве

программиста по MASM (MASMProgGuide.pdf)

Спецификации и особенности использования функций см. в справочной системе

MSDN Visual Studio.

Предполагается, что конечный вариант программы можно получить, компонуя ее

из фрагментов программ-примеров, аналогично puts_gets_puts.asm

(puts_ReadFile_MessageBox.asm) с заменой функций на заданные в варианте.

Обратите внимание: программа должна корректно работать при вводе пустой строки.

При работе понадобиться добавить в файл in_out.inc прототипы необходимых функций.

Их следует добавлять на основании спецификаций в справке MSDN или .h-файлов Visual Studio.

5. Ход работы (порядок выполнения работы)

В среде операционной системы Windows XP с использованием макроассемблера Microsoft Macro Assembler Version 7 / 8 разработать, отладить и провести обработку контрольных примеров.

Для всех задач и используемых нетривиальных процедур/функций разработать и решить контрольные программы/примеры, результаты выполнения которых очевидны или легко проверяются.

Сохранить результаты работы (решение основных и контрольных задач) в текстовом файле или в документе Word.

Составить отчет о выполнении работы.

6. Содержание отчета

Отчет должен содержать:

1. Титульный лист;
2. Формулировку цели и задач работы;

3. Индивидуальное задание на работу;
4. Описание использованных программных и аппаратных средств для выполнения работы;
5. Описание хода работы с указанием этапов и пояснениями используемых решений (методов, программ, процедур, библиотек);
6. Список использованных источников.

7. Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.
2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.
3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.

ЛАБОРАТОРНАЯ РАБОТА № 5

"Разработка программы обработки строки символов
с использованием процедур"

1. Цель и задачи работы

Освоить средства и приемы разработки программ обработки строк символов с использованием процедур.

2. Общие положения (теоретические сведения)

Пример программы обработки строк символов с использованием процедур.

```
        .686
        .MODEL    FLAT,C
        INCLUDE   in_out.inc
        OPTION PROLOGUE:none
        OPTION EPILOGUE:none

        .STACK

        .DATA
NN      =       200
LF      =       0Ah
;X0    BYTE        "321Какая-то 5 строка 78 с цифрами 123",0
Prompt      BYTE        "Введите строку:",0
Msg1 BYTE       "Введена строка:       [%s]",10,0
Msg2 BYTE       "Отсортированная строка:[%s]",0

        .DATA?
X       BYTE        NN DUP (?)
Y       BYTE        NN DUP (?)

        .CODE

;--input-----------------------------------------------
input  PROC
        push   ebp
        mov    ebp,esp
        sub    esp,4
        push   ebx

;get stdin
        call    __iob_func  ;get stdin
        mov    [ebp-4],eax
```

```
;input
        mov   ebx,[ebp+8] ;start of s
;repeating while not LF
@@:  push  [ebp-4]
        call   fgetc          ;get character from stdin
        add    esp,4
        cmp    al,LF
        je     @F             ;character is LF !
        mov    [ebx],al       ;moving character to RAM (to s)
        inc    ebx            ;next free byte
        jmp    @B             ;to next character input
@@:
        mov    BYTE PTR [ebx],0
        mov    eax,ebx
        sub    eax,[ebp+8] ; length of s

        pop    ebx
        mov    esp,ebp
        pop    ebp
        ret    4
input  ENDP

;--sort-----------------------------------------------------
; преобразование строки
sort    PROC
        push   ebp
        mov    ebp,esp
        sub    esp,4
        push   eax
        push   ecx
        push   esi
        push   edi

;get length -> n
        push   [ebp+12]
        call   strlen
        add    esp,4
        mov    ecx,eax

;2. (strcopy a -> b)
        inc    ecx
        cld
        mov    esi,[ebp+12]
        mov    edi,[ebp+8]
        rep    movsb
```

```
;3. (test n)
        cmp   eax,1
        jbe   exit
;4.
        mov   ecx,eax
        dec   DWORD PTR [ebp+8]
        dec   ecx           ; ecx:=n-1
;5.
loop_i:     mov   [ebp-4],ecx  ; for i:=1 to n-1
        mov   esi,[ebp+8]  ; [esi]:=b0 (k=0) нулевой! символ в строке
        mov   al,0          ; b0:=0
        mov   edx,0         ; flag:=0

        mov   edi,esi
        inc   ecx           ; cx:=n-i+1
;6.
loop_j:     inc   edi             ; j:=j+1
        mov   ah,[edi]      ; ah:=s[j]
;7.
        cmp   ah,'0'        ; ah - digit ? ...
        jb    continue_j    ; not digit !
        cmp   ah,'9'
        ja    continue_j    ; not digit !

;8. (найдена очередная цифра)
       cmp    ah,al         ; b[j] <> b[k] ?  (сравнение)
        jae   continue      ; b[j] >= b[k] !  (если порядок правильный)
        mov   [esi],ah      ; b[j] <-> b[k]   (если порядок неправильный)
        mov   [edi],al      ; b[j] <-> b[k]
        mov   edx,1         ; flag:=1
;9.
continue:                   ; порядок следования цифр правильный!
        mov   esi,edi             ; k:=j
        mov   al,[esi]      ; k:=j
;10, 11.
continue_j:
        loop  loop_j              ; j:=j+1, если j<n, 6.
;12.
        test  edx,edx
        je    exit
;13, 14.
        mov   ecx,[ebp-4]
        loop  loop_i              ; i:=i+1, если i<n, 5.
;15.
```

```
exit:
        pop     edi
        pop     esi
        pop     ecx
        pop     eax
        mov     esp,ebp
        pop     ebp
        ret     8
sort    ENDP

;--main-------------------------------------------------
main  PROC
        push    ebp
        mov     ebp,esp
;       int     3
;input
        push    OFFSET Prompt
        call    printf
        add     esp,4
        push    OFFSET X
        call    input
;processing
        push    OFFSET X
        push    OFFSET Y
        call    sort
;output
        push    OFFSET X
        push    OFFSET Msg1
        call    printf
        add     esp,8
        push    OFFSET Y
        push    OFFSET Msg2
        call    printf
        add     esp,8

        leave
        ret

main  ENDP

        END ;program
```

---

lab05_ex.asm

```
        .686
        .MODEL    FLAT,C
        INCLUDE   in_out.inc
input   PROTO     PASCAL :PTR BYTE
sort    PROTO     PASCAL :PTR BYTE, :PTR BYTE

        .STACK

        .DATA
NN      =       200
LF      =       0Ah
;X0    BYTE        "321Какая-то 5 строка 78 с цифрами 123",0
Prompt      BYTE        "Введите строку:",0
Msg1 BYTE       "Введена строка:      [%s]",10,0
Msg2 BYTE       "Отсортированная строка:[%s]",0

        .DATA?
X       BYTE        NN DUP (?)
Y       BYTE        NN DUP (?)

        .CODE

;--input-----------------------------------------------------
input   PROC        PASCAL USES ebx s:PTR BYTE
        LOCAL       stdin:DWORD

;get stdin
        INVOKE    __iob_func  ;get stdin
        mov   stdin,eax
;input
        mov   ebx,s         ;start of s
;repeating while not LF
@@:  INVOKE    fgetc,stdin   ;get character from stdin
        cmp   al,LF
        je    @F            ;character is LF !
        mov   [ebx],al      ;moving character to RAM (to s)
        inc   ebx           ;next free byte
        jmp   @B            ;to next character input
@@:
        mov   BYTE PTR [ebx],0
        mov   eax,ebx
        sub   eax,s         ; length of s

        RET
input   ENDP
```

```
;--sort-----------------------------------------------------
; преобразование строки
sort    PROC        PASCAL USES eax ecx esi edi a:PTR BYTE, b:PTR BYTE
        LOCAL       i_:DWORD

;get length -> n
        INVOKE    strlen,a
        mov   ecx,eax

;2. (strcopy a -> b)
        inc     ecx
        cld
        mov   esi,a
        mov   edi,b
        rep     movsb
;3. (test n)
        mov   ecx,eax
        cmp   ecx,1
        jbe     exit
;4.
        dec     b
        dec     ecx             ; ecx:=n-1
;5.
loop_i:     mov   i_,ecx            ; for i:=1 to n-1
        mov   esi,b             ; [esi]:=b0 (k=0) нулевой! символ в строке
        mov   al,0              ; b0:=0
        mov   edx,0             ; flag:=0

        mov   edi,esi
        inc     ecx             ; cx:=n-i+1
;6.
loop_j:     inc     edi                 ; j:=j+1
        mov   ah,[edi]          ; ah:=b[j]
;7.
        cmp   ah,'0'            ; ah - digit ? ...
        jb      continue_j      ; not digit !
        cmp   ah,'9'
        ja      continue_j      ; not digit !

;8.  найдена очередная цифра
       cmp     ah,al            ; b[j] <> b[k] ?  (сравнение)
        jae     continue        ; b[j] >= b[k] !  (если порядок правильный)
        mov   [esi],ah          ; b[j] <-> b[k]   (если порядок неправильный)
        mov   [edi],al          ; b[j] <-> b[k]
```

```
        mov   edx,1         ; flag:=1
;9.
continue:                   ; порядок следования цифр правильный!
        mov   esi,edi               ; k:=j
        mov   al,[esi]      ; k:=j
;10, 11.
continue_j:
        loop  loop_j                ; j:=j+1, если j<n, 6.
;12.
        test  edx,edx
        je    exit
;13, 14.
        mov   ecx,i_
        loop  loop_i                ; i:=i+1, если i<n, 5.
;15
exit:   RET
sort    ENDP

;--main-------------------------------------------------
main  PROC
;       int     3
;input
        INVOKE    printf,ADDR Prompt
        INVOKE    input,ADDR X
;processing
        INVOKE    sort,ADDR X,ADDR Y
;output
        INVOKE    printf,ADDR Msg1,ADDR X
        INVOKE    printf,ADDR Msg2,ADDR Y

        RET
main  ENDP

        END ;program
```

```
D:\Lab_05>lab05_ex.exe
Введите строку:123
Введена строка:       [123]
Отсортированная строка:[123]

D:\Lab_05>lab05_ex.exe
Введите строку:321
```

```
Введена строка:       [321]
Отсортированная строка:[123]

D:\Lab_05>lab05_ex.exe
Введите строку:21
Введена строка:       [21]
Отсортированная строка:[12]

D:\Lab_05>lab05_ex.exe
Введите строку:qdhdrfj3094421urjwjk1391-2f349r1-3492k
Введена строка:       [qdhdrfj3094421urjwjk1391-2f349r1-3492k]
Отсортированная строка:[qdhdrfj0111122urjwjk2333-3f444r4-9999k]

D:\Lab_05>lab05_ex.exe
Введите строку:jwr934301=42491131932844З4i02iwjd2rmmf
Введена строка:       [jwr934301=42491131932844З4i02iwjd2rmmf]
Отсортированная строка:[jwr001111=22223333344444448i99iwjd9rmmf]

D:\Lab_05>lab05_ex.exe
Введите строку:
Введена строка:       []
Отсортированная строка:[]

----------------------------
D:\Lab_05>lab05_ex2.exe
Введите строку:123
Введена строка:       [123]
Отсортированная строка:[123]

D:\Lab_05>lab05_ex2.exe
Введите строку:321
Введена строка:       [321]
Отсортированная строка:[123]

D:\Lab_05>lab05_ex2.exe
Введите строку:21
Введена строка:       [21]
Отсортированная строка:[12]
```

3. Объекты исследования, оборудование, материалы и наглядные пособия

Объект исследования – средства и приемы разработки программ обработки строк символов с использованием процедур.

В качестве оборудования используются персональные компьютеры учебных классов кафедры ПМиИ (ауд. 12-207, 12-209, 12-211).

В качестве операционной системы используется операционная система MS Windows XP SP2.

Среда разработки – макроассемблер "Microsoft Macro Assembler Version 7 / 8".

Средства ввода: клавиатура или текстовый файл.

Средства ввода: экран ПК или текстовый файл.

4. Задание на работу (рабочее задание)

Разработать программу, которая вводит строку X, выполняет ее преобразование и выводит исходную строку X и результат преобразования – строку Y.

Диалог работы с программой должен выглядеть приблизительно так:

>prog.exe[Enter]

Введите строку:X[Enter]

Введена строка:X

Строка-результат:Y

Для выполнения ввода и вывода нужно использовать те же функции, что и в индивидуальном задании к ЛАБОРАТОРНАЯ РАБОТА № 4. Если функция ввода/вывода является посимвольной, то на ее основе следует сделать внутреннюю процедуру так, чтобы соответствующая операция ввода/вывода выполнялась с помощью вызова процедуры (см. процедуру input в примере lab05_ex.asm). Процедура должна иметь параметр, передаваемый через стек – адрес вводимой/выводимой строки.

Преобразование строки X в строку Y следует выполнять тоже с помощью процедуры (нескольких процедур). Основная процедура в этой группе процедур должна иметь 2 параметра – адреса строк X и Y, передаваемые через стек. Для процедуры преобразования строки X в строку Y разработать алгоритм в виде нумерованного перечня шагов на псевдокоде (пример см. в приложении) или в виде блок-схемы. Детализация алгоритма должна быть достаточной для построения программы на универсальном языке программирования высокого уровня (таком как C или Pascal). Реализовать алгоритм в виде процедуры на языке ассемблера.

Таким образом, программа должна содержать минимум 3 процедуры: главную – main; процедуру ввода (вывода) (в примере – input); процедуру преобразования строки X в строку Y (в примере – sort).

По крайней мере, в одной процедуре должны использоваться локальные переменные в стеке. Во всех процедурах явно используемые регистры должны восстанавливать свои значения до вызова процедуры (кроме регистра eax, если он используется в качестве возвращаемого значения процедуры-функции).

Программа должна быть реализована в двух вариантах.

В первом варианте для написания и вызова процедур надо максимально использовать возможности директив PROTO, PROC, LOCAL, RET, INVOKE и автоматически формируемых пролога и эпилога процедур так, чтобы:

– обращение к параметрам процедур и локальным переменным в стеке выполнялось с помощью символических имен;
– пролог, включающий в себя сохранение и установку регистра ebp, резервирование места под локальные переменные и сохранение в стеке используемых регистров, формировался автоматически;
– эпилог, включающий в себя восстановление из стека используемых регистров, освобождение места под локальные переменные и восстановление регистра ebp, возврат управления из процедуры с автоматической очисткой стека от параметров формировался по директиве RET;
– все вызовы процедур выполнялись с помощью директивы INVOKE.

Во втором варианте для написания и вызова процедур надо минимально использовать возможности директивы PROC; для локальных процедур не использовать прототипы; не использовать директивы LOCAL, RET, INVOKE и автоматически формируемые прологи и эпилоги процедур так, чтобы:

– обращение к параметрам процедур и локальным переменным в стеке выполнялось с помощью косвенной адресации с использованием регистра ebp;
– пролог, включающий в себя сохранение и установку регистра ebp, резервирование места под локальные переменные и сохранение в стеке используемых регистров, формировался явно командами процессора;
– эпилог, включающий в себя восстановление из стека используемых регистров, освобождение места под локальные переменные и восстановление регистра ebp, возврат управления из процедуры с автоматической очисткой стека от параметров формировался явно командами процессора;
– все вызовы процедур выполнялись с помощью команды процессора call (если нужно, то параметры из стека удалять явно).

5. Ход работы (порядок выполнения работы)

В среде операционной системы Windows XP с использованием макроассемблера Microsoft Macro Assembler Version 7 / 8 разработать, отладить и провести обработку контрольных примеров.

Для всех задач и используемых нетривиальных процедур/функций разработать и решить контрольные программы/примеры, результаты выполнения которых очевидны или легко проверяются.

Сохранить результаты работы (решение основных и контрольных задач) в текстовом файле или в документе Word.

Составить отчет о выполнении работы.

6. Содержание отчета

Отчет должен содержать:

1. Титульный лист;
2. Формулировку цели и задач работы;
3. Индивидуальное задание на работу;
4. Описание использованных программных и аппаратных средств для выполнения работы;
5. Описание хода работы с указанием этапов и пояснениями используемых решений (методов, программ, процедур, библиотек);
6. Список использованных источников.

7. Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.
2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.
3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.

ЛАБОРАТОРНАЯ РАБОТА № 6

"Разработка программы обработка массива целых чисел"

1. Цель и задачи работы

Освоить средства и приемы разработка программ обработка массива целых чисел.

2. Общие положения (теоретические сведения)

Пример программы обработка массива целых чисел.

```
        .686
        .MODEL    FLAT,C
        INCLUDE   in_out.inc

        .STACK

        .DATA
NN      =       200
NNs     =       2000

Prompt      BYTE        "Введите массив %s:",0
msg     BYTE        "Массив %s:",0
Title_A BYTE        "A",0
Title_B BYTE        "B",0
Title_C BYTE        "C",0
i_form      BYTE        "%d",0
o_form      BYTE        "%4d",0
o_LF    BYTE        0Ah,0


        .DATA?
buff    BYTE        NNs DUP (?)
AA      SWORD       NN DUP (?)
BB      SWORD       NN DUP (?)
CC      SWORD       NN DUP (?)
m       DWORD       ?
n       DWORD       ?

        .CODE

;--input-------------------------------------------------
input   PROC        PASCAL USES ebx esi tit:PTR BYTE, A:PTR SWORD
        LOCAL       x:SDWORD,i:PTR BYTE,m_:DWORD
```

```
;input string
        mov   esi,A
        INVOKE    printf,ADDR Prompt,tit
        INVOKE    gets,ADDR buff+1
        INVOKE    strlen,ADDR buff
        mov   WORD PTR buff[eax],0020h
        mov   m_,0
        mov   i,OFFSET buff

;parsing of string
loop1:
        INVOKE    sscanf,i,ADDR i_form,ADDR x
        cmp   eax,1
        jne   exit
        inc   m_
        mov   eax,x                 ; convertion to SWORD
        mov   SWORD PTR [esi],ax    ; and A[m]=x
        add   esi,2
        mov   ebx,i
@@: inc   ebx
        cmp   BYTE PTR [ebx],' '        ; compare with space
        je    @B
@@: inc   ebx
        cmp   BYTE PTR [ebx],' '        ; compare with space
        jne   @B
        mov   i,ebx
        jmp   loop1

exit:
        mov   eax,m_
        RET
input  ENDP

;--output-------------------------------------------------
output      PROC      PASCAL USES eax ebx ecx tit:PTR BYTE, A:PTR
SWORD, m_:DWORD
        LOCAL     x:SDWORD,n_:DWORD

;output title
        INVOKE    printf,ADDR msg,tit

;output elements
        mov   ebx,A
        mov   ecx,m_
        cmp   ecx,0
```

```
        je      exit

@@:     mov     n_,ecx
        mov     ax,[ebx]
        cwde                    ; expansion to DWORD
        mov     x,eax
        INVOKE  printf,ADDR o_form,x
        add     ebx,2
        mov     ecx,n_
        loop    @B
exit:
        INVOKE  printf,ADDR o_LF
        RET
output  ENDP

;--change- (insert zeros left) ------------------------------
change  PROC    PASCAL USES ecx esi edi A:PTR SWORD,
m_:PTR DWORD, n_:DWORD

;moving elements to right
        mov     esi,m_
        mov     ecx,[esi]
        mov     esi,ecx
        dec     esi
        shl     esi,1
        add     esi,A
        mov     edi,n_
        dec     edi
        shl     edi,1
        add     edi,A
        std
        rep     movsw
        cld

;make zeros
        mov     ecx,n_
        mov     esi,m_
        sub     ecx,[esi]
@@:     mov     SWORD PTR [edi],0
        sub     edi,2
        loop    @B
exit:
        mov     esi,m_
        mov     ecx,n_
        mov     [esi],ecx
```

```
        RET
change  ENDP

;--change1-- make zero right -------------------------------
change1 PROC    PASCAL USES ecx esi edi A:PTR SWORD,
m_:PTR DWORD, n_:DWORD

;make zeros
        mov   ecx,n_
        mov   esi,m_
        sub   ecx,[esi]
        je    exit
        mov   edi,[esi]
        shl   edi,1
        add   edi,A
@@:     mov   SWORD PTR [edi],0
        add   edi,2
        loop  @B
exit:
        mov   esi,m_
        mov   ecx,n_
        mov   [esi],ecx
        RET
change1 ENDP

;--sort---------------------------------------------------
; insertion method
sort    PROC    PASCAL USES eax ebx ecx esi edi A:PTR SWORD,
m_:DWORD

;test of length
        mov   ecx,m_
        cmp   ecx,1
        jbe   exit

        dec   ecx           ; ecx:=m-1
        mov   esi,A
loop_i: add   esi,2
        mov   ax,[esi]      ; t:=A[i]
        mov   edi,esi       ; j:=i
@@:     sub   edi,2         ; j:=j-1
        cmp   edi,A         ; j cmp 1
        jb    @F
        cmp   [edi],ax
        jle   @F
```

```
        mov   bx,[edi]      ; A[j+1]:=A[j]
        mov   [edi+2],bx
        jmp   @B
@@:     mov   [edi+2],ax    ; A[j+1]:=t
        loop  loop_i
exit:   RET
sort    ENDP

;--make_C-----------------------------------------------
; C[i]:=4*i-(A[i]+B[i])/3+B[i]
make_C      PROC        PASCAL USES eax ebx ecx esi edi A:PTR SWORD,
B:PTR SWORD, C_:PTR SWORD, m_:DWORD
        LOCAL       x3:SWORD

;test of length
        mov   ecx,m_            ; i=ecx:=m
        cmp   ecx,1
        jb    exit

        mov   x3,3
        mov   esi,A
        mov   edi,B
        mov   ebx,C_
        mov   eax,ecx
        shl   eax,1
        add   esi,eax           ; esi=A+m*2
        add   edi,eax           ; edi=B+m*2
        add   ebx,eax           ; ebx=C+m*2

@@:     sub   esi,2         ; A[i]=[esi]
        sub   edi,2         ; B[i]=[edi]
        sub   ebx,2         ; C[i]=[ebx]
        mov   ax,[esi]      ; ax:=A[i]
        add   ax,[edi]      ; ax:=A[i]+B[i]
        xor   dx,dx         ; dx:=0000 (регистр dx должен быть заполнен зна-
ковым разрядом регистра ax)
        cmp   ax,0
        jge   continue
        not   dx            ; dx:=FFFF (регистр dx должен быть заполнен зна-
ковым разрядом регистра ax)
continue:
        idiv  x3            ; ax:=[dx:ax]/3=(A[i]+B[i])/3
        sub   ax,[edi]      ; ax:=(A[i]+B[i])/3-B[i]
        mov   dx,cx         ; dx:=i
        shl   dx,2          ; dx:=4*i
```

```
        sub   dx,ax         ; dx:=dx-ax=4*i-(A[i]+B[i])/3+B[i]
        mov   [ebx],dx      ; C[i]:=dx
        loop  @B            ; i:=i-1

exit:   RET
make_C  ENDP

;--main-------------------------------------------------
main  PROC
;       int     3
;input
        mov   buff,' '; set first character of buffer -> ' '
        INVOKE    input,ADDR Title_A,ADDR AA
        mov   m,eax
        INVOKE    input,ADDR Title_B,ADDR BB
        mov   n,eax

;change & sorting
        cmp   eax,m
        je      sort_B
        ja      @F
        INVOKE    change,ADDR BB, ADDR n, m
        INVOKE    sort,ADDR AA, m
        jmp   processing
@@: INVOKE    change,ADDR AA, ADDR m, n
sort_B:       INVOKE    sort,ADDR BB, n

processing:
        INVOKE    make_C,ADDR AA,ADDR BB,ADDR CC, m

;output result
        INVOKE    output,ADDR Title_A,ADDR AA, m
        INVOKE    output,ADDR Title_B,ADDR BB, n
        INVOKE    output,ADDR Title_C,ADDR CC, m

        RET
main  ENDP

        END  ;program
```

3. Объекты исследования, оборудование, материалы и наглядные пособия

Объект исследования – разработка программы обработка массива целых чисел.

В качестве оборудования используются персональные компьютеры учебных классов кафедры ПМиИ (ауд. 12-207, 12-209, 12-211).

В качестве операционной системы используется операционная система MS Windows XP SP2.

Среда разработки – макроассемблер "Microsoft Macro Assembler Version 7 / 8".

Средства ввода: клавиатура или текстовый файл.

Средства ввода: экран ПК или текстовый файл.

4. Задание на работу (рабочее задание)

Разработать программу, которая вводит два массива целых чисел – A и B – и формирует по ним новый массив – C (предполагается, что число элементов массива A равно m, массива B – n; при этом $m \geq 0$, $n \geq 0$).

Число элементов нового массива C – k – определяется по правилу k=min(m,n) или k=max(m,n). При этом один из исходных массивов (A или B) может изменить свою размерность! Другой, у которого не меняется число элементов, перед формированием C должен быть отсортирован (если n=m, то следует сортировать B).

Правила определения числа k элементов нового массива C, согласования числа элементов в массивах A, B и расчета элементов массива C определяются в индивидуальном задании.

В параметрах индивидуального задания указываются:

7. тип элементов массивов A, B, C: BYTE, SBYTE, WORD, SWORD, DWORD или SDWORD;
8. правило определения числа элементов k массива C: k=min(m,n) или k=max(m,n);
9. область усечения (при k=min(m,n)) / расширения (при k=max(m,n)) большего / меньшего (по количеству элементов) массива: слева или справа;
10. порядок сортировки массива, в котором не меняется число элементов;
11. правило получения C[i]:=f(i,A[i],B[i]) (i=1,2,...,k) (здесь используются массивы A и B с уже измененным (если нужно) числом элементов – k).

Примечания:

к п. 1]: типы BYTE, WORD, DWORD предполагают использования чисел без знака (только неотрицательных); типы SBYTE, SWORD, SDWORD соответствуют числам со знаком формата байт, слово и двойное слово соответственно;

к п. 3]: для согласования размерности при k=min(m,n) из большего по количеству элементов массива A или B исключаются начальные элементы (при усечении слева) или последние элементы (при усечении справа); при k=max(m,n) в меньший по количеству элементов массив A или B добавляются нулевые элементы – в начало при расширении слева или – в конец при расширении справа;

к п. 5]: в f(x) деления выполнять нацело; если при делении знаменатель d оказывается равным 0, то полагать d=1; выражение mod(x,y) обозначает остаток

от деления x/y (считать, что mod(x,0)=0); выражение x^m обозначает степень вида x^m;

к работе в целом: эта программа будет основой при формировании контрольных заданий на зачете; по заданию преподавателя надо будет внести изменения в программу в соответствии с изменением задания в одном пункте: 2], 3], 4] или 5]; скомпилировать и продемонстрировать измененный вариант программы.

Указания:

1) Должны быть разработаны процедуры (с параметрами в стеке!):

– input(promt, address) – процедура ввода, возвращающая в eax число введенных элементов:
  promt – адрес строки запроса, address – адрес массива;
– change(address,*m,*n) – процедура модификации одного массива (m->n):
  address – адрес массива, – *m адрес исходной размерности, *n – адрес размерности результата;
– sort(address,m) – процедура сортировки массива:
  address – адрес массива, m – число элементов массива;
– make_C(*A,*B,*C,k) – процедура формирования массива C:
  *A, *B, *C – адреса массивов A, B, C соответственно; k – число элементов нового массива C;
– output(message,address,k) – процедура вывода массива:
  message – адрес начала фрагмента строки-сообщения, индивидуализирующего выводимый массив; address – адрес массива; k – число элементов массива.

2) Ввод в процедуре input следует выполнять с помощью CRT-функций gets и sscanf так, чтобы вводимые числа можно было набирать в одной строке через пробел. Число чисел не вводится отдельно, а подсчитывается как число введенных чисел.

3) Вывод в процедуре output выполнять с помощью CRT-функции fprintf.

4) Умножение на 2, 4, 8 выполнять с помощью арифметических команд сдвига влево sal

5) Деление на 2, 4, 8 выполнять с помощью арифметических команд сдвига вправо sar

5. Ход работы (порядок выполнения работы)

В среде операционной системы Windows XP с использованием макроассемблера Microsoft Macro Assembler Version 7 / 8 разработать, отладить и провести обработку контрольных примеров.

Для всех задач и используемых нетривиальных процедур/функций разработать и решить контрольные программы/примеры, результаты выполнения которых очевидны или легко проверяются.

Сохранить результаты работы (решение основных и контрольных задач) в текстовом файле или в документе Word.

Составить отчет о выполнении работы.

6. Содержание отчета

Отчет должен содержать:

1. Титульный лист;
2. Формулировку цели и задач работы;
3. Индивидуальное задание на работу;
4. Описание использованных программных и аппаратных средств для выполнения работы;
5. Описание хода работы с указанием этапов и пояснениями используемых решений (методов, программ, процедур, библиотек);
6. Список использованных источников.

7. Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.
2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.
3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.

ЛАБОРАТОРНАЯ РАБОТА № 7

"Разработка программы обработки вещественных чисел
с использованием математического сопроцессора"

1. Цель и задачи работы

Освоить средства и приемы разработки программ обработки вещественных чисел с использованием математического сопроцессора.

2. Общие положения (теоретические сведения)

Примеры использования обработки вещественных чисел с использованием математического сопроцессора.

Nonarithmetic Instructions	Arithmetic Instructions
FABS	F2XM1
FCHS	FADD (P)
FCLEX	FBLD
FDECSTP	FBSTP
FFREE	FCOMP(P)(P)
FINCSTP	FCOS
FINIT	FDIV(R)(P)
FLD (register-to-register)	FIADD
FLD (extended format from memory)	FICOM(P)
FLD constant	FIDIV(R)
FLDCW	FILD
FLDENV	FIMUL
FNOP	FIST(P)
FRSTOR	FISUB(R)
FSAVE	FLD (conversion)
FST(P) (register-to-register)	FMUL(P)
FSTP (extended format to memory)	FPATAN
FSTCW	FPREM
FSTENV	FPREM1
FSTSW	FPTAN
FWAIT	FRNDINT
FXAM	FSCALE
FXCH	FSIN
	FSINCOS
	FSQRT
	FST(P) (conversion)
	FSUB(R)(P)
	FTST

```
FUCOM(P)(P)
FXTRACT
FYL2X
FYL2XP1
```

FINIT

FLD (extended format from memory)

```
fld     z       ; DQ
fld     fpu_2 ; DQ 2.0
```

можно загружать фиксированные константы (- специальные команды)

FSTP (extended format to memory)

```
fstp    z
```

Фрагмент умножения z на 2.0 (z=z*2)
(инициализация finit не обязательна, но рекомендуется)

```
fld     z
fld     fpu_2 ; 2
fmul
fstp    z
```

3. Объекты исследования, оборудование, материалы и наглядные пособия

Объект исследования – средства и приемы разработки программ обработки вещественных чисел с использованием математического сопроцессора.

В качестве оборудования используются персональные компьютеры учебных классов кафедры ПМиИ (ауд. 12-207, 12-209, 12-211).

В качестве операционной системы используется операционная система MS Windows XP SP2.

Среда разработки – макроассемблер "Microsoft Macro Assembler Version 7 / 8".

Средства ввода: клавиатура или текстовый файл.

Средства ввода: экран ПК или текстовый файл.

4. Задание на работу (рабочее задание)

Разработать программу, которая вводит два массива вещественных чисел – A и B – и формирует по ним новый массив – C (предполагается, что число элементов массива A равно m, массива B – n; при этом $m \geq 0$, $n \geq 0$).

Число элементов нового массива C – k – определяется по правилу k=min(m,n) или k=max(m,n). При этом один из исходных массивов (A или B) может изменить свою размерность! Другой, у которого не меняется число эле-

ментов, перед формированием C должен быть отсортирован (если n=m, то следует сортировать B).

Правила определения числа k элементов нового массива C, согласования числа элементов в массивах A, B и расчета элементов массива C определяются в индивидуальном задании.

В параметрах индивидуального задания указываются:

12. тип элементов массивов A, B, C: DWORD, QDWORD, TBYTE;
13. правило определения числа элементов k массива C: k=min(m,n) или k=max(m,n);
14. область усечения (при k=min(m,n)) / расширения (при k=max(m,n)) большего / меньшего (по количеству элементов) массива: слева или справа;
15. порядок сортировки массива, в котором не меняется число элементов;
16. правило получения C[i]:=f(i,A[i],B[i]) (i=1,2,...,k) (здесь используются массивы A и B с уже измененным (если нужно) числом элементов – k).

5. Ход работы (порядок выполнения работы)

В среде операционной системы Windows XP с использованием макроассемблера Microsoft Macro Assembler Version 7 / 8 разработать, отладить и провести обработку контрольных примеров.

Для всех задач и используемых нетривиальных процедур/функций разработать и решить контрольные программы/примеры, результаты выполнения которых очевидны или легко проверяются.

Сохранить результаты работы (решение основных и контрольных задач) в текстовом файле или в документе Word.

Составить отчет о выполнении работы.

6. Содержание отчета

Отчет должен содержать:

1. Титульный лист;
2. Формулировку цели и задач работы;
3. Индивидуальное задание на работу;
4. Описание использованных программных и аппаратных средств для выполнения работы;
5. Описание хода работы с указанием этапов и пояснениями используемых решений (методов, программ, процедур, библиотек);
6. Список использованных источников.

7. Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.

2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.
3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.

ЛАБОРАТОРНАЯ РАБОТА № 8

"Разработка программы с обработкой прерываний и исключений сопроцессора"

1. Цель и задачи работы

Освоить средства и приемы разработки программ с обработкой прерываний и исключений сопроцессора.

2. Общие положения (теоретические сведения)

Организация прерывания процессора

В процессе выполнения любой программы, как внутри ЭВМ, так и во внешней среде (машины используемые для управления некоторыми объектами), могут возникать некоторые события, которые требуют немедленной, определенной для этого события реакции, как со стороны ЭВМ, так и со стороны программы. ВМ временно прекращают выполнение текущей программы, переходит к выполнению некоторой другой программы, специально предусмотренной для этого случая, а по завершении специальной программы ЭВМ

возвращается к выполнению исходной программы. Такой процесс перехода к специальным программам и обратно носит названия прерывание, причем исходная программа называется прерванной, а специальная программа - прерывающая. Различного рода сигналы, которые информируют ЭВМ о наступлении события, носят название запроса прерывания. Любое прерывание текущей программы заключается в том, что:

1) Процессор прекращает ее выполнение
2) Запоминает информацию, нужную для продолжения выполнения программы с точки прерывания.
3) Переходит к выполнению специальной программы.
4) После исполнения специальной программы, управление возвращается исходной программе.

Запросы на прерывание могут возникнуть внутри компьютера (сбой в некотором устройстве, переполнение разрядной сетки, попытка деления на нуль, требование прерываний ввода-вывода и т.п.). Несмотря на программу источник, временной момент появления прерывания определить не возможно. То есть запросы прерывания асинхронны по отношению к программе. Моменты возникновения других событий можно заранее предсказать. При многократном использовании программы эти события будут возникать в одно время (операции обмена, переполнения) - синхронные события. Самые сложные события асинхронные. Для организации правильной работы основной и прерывающей программы нужно управление. Эта координация реализуется системами прерывания. Основными функциями являются следующие:

1) временной останов выполняющей программы и выбор запроса на обслуживание
2) запоминание состояния прерванной программы
3) инициирование программ - обработчиков прерывания
4) обслуживание - выполнение прерывающей программы
5) восстановление состояния прерывающей программы и возврат к выполнению исходной программы

Эти функции также называют последовательностью обработки любого запроса на прерывание.

Любой запрос на прерывание формируется по соответствующей причине. Обычно: IR [0:n] -регистр прерывания. От момента выработки запроса до момента прерывания существует некоторое время, время реакции системы прерывания. Любой запрос фиксируется в фиксированном бите слова регистра прерывания. Время реакции в общем случае не постоянно. Оно зависит не только от того, как выпускаются допустимые моменты прерывания, но и от того, Сколько программ с более высоким приоритетом, чем прерывание ждет своего обслуживание в очереди. В ЭВМ используются различные способы определения допустимого момента прерывания. Самый простой способ заключается в том, что в формате команды ЭВМ вводится специальный бит - признак разрешения прерывания.

Таким образом, программист, составляя программу, может управлять разрешением прерывания. Это позволяет минимизировать объем информации, который запасается при переходе к прерывающей программе, что уменьшает общее время обработки прерывания, но само время реакции оказывается достаточно большим. Более распространенный способ предполагает, что прерывание возможно после окончания любой текущей команды. Но в этом случае нужно сохранять содержание всех программно доступных регистров. Это уменьшает время реакции, но увеличивает накладные расходы. В этом случае время реакции системы прерывания не превышает длительности выполнения самой длинной команды. Существует еще третий вариант - для машин, работающих в реальном времени. В таких ЭВМ прерывание может допускаться на любом шаге выполнения команды. Это характерно для компьютеров, имеющих микропрограммный уровень. Время реакции сводится к длительности одного такта. Но объем запоминаемой информации требует значительных временных затрат.

Выбор запроса на обслуживание - вторая часть задачи. Запросы поступают в любой момент времени, следовательно, к моменту, когда может быть разрешено прерывание в регистре запроса может находится несколько запросов (по крайней мере два). Возникает задача выбора запроса на обслуживание. Любой запрос в компьютере снабжается своим приоритетом. Обычно в качестве приоритета берут 0,1,...n. Степень важности обработчика пропорционален номеру (0 - самый высокий приоритет). Порядок выбора запросов определяется дисциплинами обслуживания. По способу исполнения приоритетов различаются дисциплины с абсолютными и относительными приоритетами. Если к моменту разрешения прерывания выбранный запрос оказывается выше приоритета текущей программы, то возможно два варианта действия:

1) текущая программа прерывается в тот момент, когда выбран приоритетный запрос (дисциплины с абсолютным приоритетом)

2) текущая программа не прерывается, продолжается до момента естественного прерывания (дисциплины с относительным приоритетом)

Независимо от вида приоритета в конкретной системе прерывания процедура обработки запроса может быть представлена так:

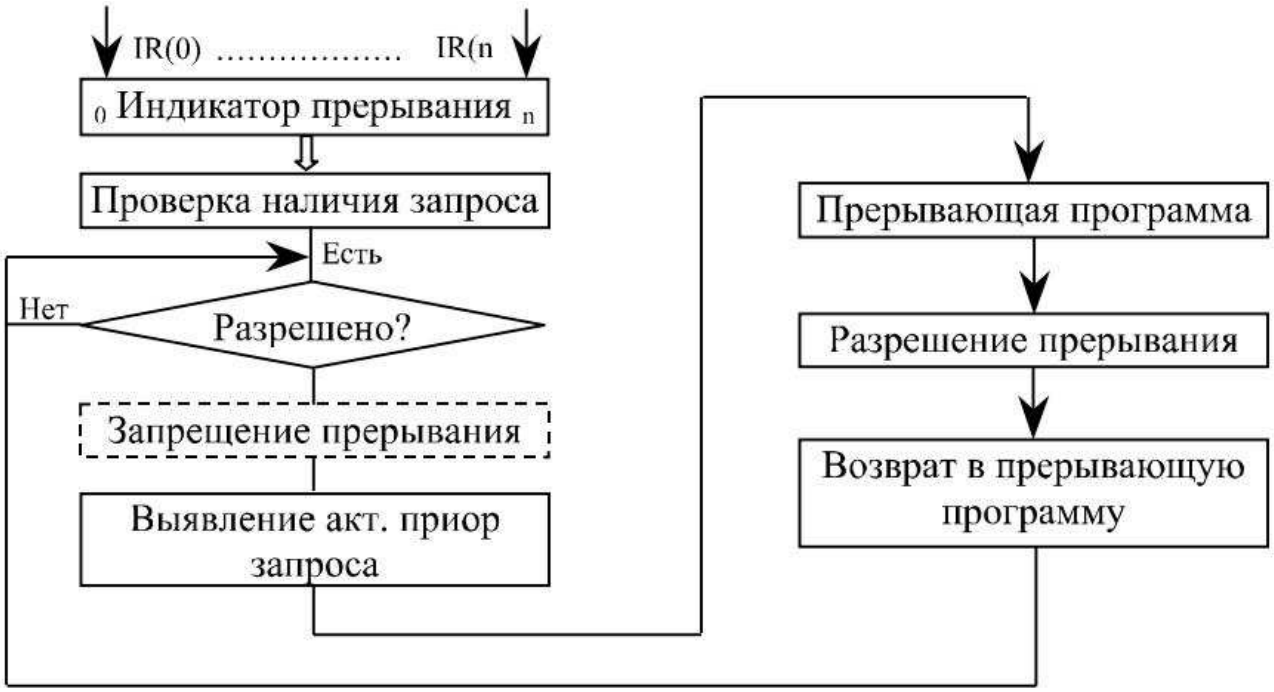


Рис. 1. Процедура обработки запроса

Функции могут реализовываться как программно, так и аппаратно. Программная реализация увеличивает время обслуживания прерывания. Максимальное время - выявление активного запроса в соответствии в соответствии с приоритетом нужно построить процедуру, которая должна из п запросов выявить приоритетный. Самый простой вариант анализа запросов - последовательный просмотр запросов слева направо (схема алгоритма - самостоятельно). Последовательный перебор характерен тем, что система прерывания имеет глубину прерывания 1 (прервать прерывающую программу нельзя). Результатом выявления активного запроса является формирование системного прерывания начального адреса прерывающей программы. В процессе работы ЭВМ важность выполняемых программ не является постоянной. Между различными прерывающими программами (запросами) не всегда можно установить фиксированное распространение приоритетов, следовательно, средства прерывания должны быть программно изменяемыми. Существует два способа программного управления приоритетами прерывающих программ:

1) Любая текущая, выполняемая в данный момент программа, характеризуется некоторым кодом (приоритетом программы). В этом случае система прерывания после выявления активного запроса сравнивает приоритет этого запроса с порогом приоритета, выполняемой программы. Если приоритет выделенного запроса ниже приоритета программы, то ее прерывание не происходит, но

поскольку приоритет программы устанавливается пользователем, то при этом, запросом при смене приоритета программы может быть прерывание. При этом аппаратных изменений никаких нет, все включается в процедуру выявления.

2) Маскирование. В систему прерывание кроме индикатора (программно не доступен) вводится регистр маскирования, который доступен программно (т.е. можно загружать в него данные). Имеет аналогичную индикатору структуру. Любой запрос индикатора поставлен в соответствующий бит маски. Взаимодействие запросов прерывания и битов маскирования строятся по простой схеме: M [i] - маска IR[i].

Если в бите маски 1, следовательно, соответствующее прерывание разрешено для обработки (для выявления), если 0, следовательно, соответствующее прерывание замаскировано (не разрешено). Изменяя маску, можем управлять приоритетом запросов на программном уровне.

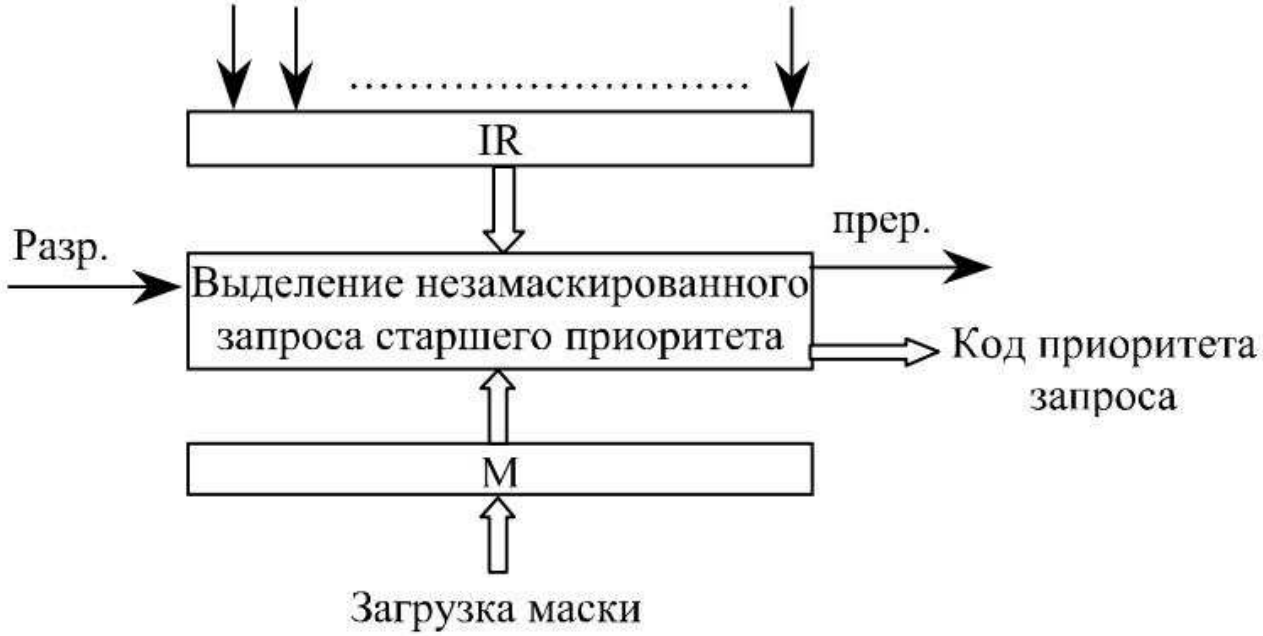


Рис. 2. Схема взаимодействия

3. Объекты исследования, оборудование, материалы и наглядные пособия

Объект исследования – средства и приемы разработки программ с обработкой прерываний и исключений сопроцессора.

В качестве оборудования используются персональные компьютеры учебных классов кафедры ПМиИ (ауд. 12-207, 12-209, 12-211).

В качестве операционной системы используется операционная система MS Windows XP SP2.

Среда разработки – макроассемблер "Microsoft Macro Assembler Version 7 / 8".

Средства ввода: клавиатура или текстовый файл.

Средства ввода: экран ПК или текстовый файл.

4. Задание на работу (рабочее задание)

Разработать две программы:

программу, выполняющую вывод в последовательный порт с использованием режима прерываний;

программу, выполняющую ввод из последовательного порта с использованием режима прерываний.

Организовать взаимодействие этих программ на двух компьютерах с использованием 0-модемного соединения по последовательному порту.

5. Ход работы (порядок выполнения работы)

В среде операционной системы Windows XP с использованием макроассемблера Microsoft Macro Assembler Version 7 / 8 разработать, отладить и провести обработку контрольных примеров.

Для всех задач и используемых нетривиальных процедур/функций разработать и решить контрольные программы/примеры, результаты выполнения которых очевидны или легко проверяются.

Сохранить результаты работы (решение основных и контрольных задач) в текстовом файле или в документе Word.

Составить отчет о выполнении работы.

6. Содержание отчета

Отчет должен содержать:

1. Титульный лист;
2. Формулировку цели и задач работы;
3. Индивидуальное задание на работу;
4. Описание использованных программных и аппаратных средств для выполнения работы;
5. Описание хода работы с указанием этапов и пояснениями используемых решений (методов, программ, процедур, библиотек);
6. Список использованных источников.

7. Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.
2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.

3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.