**Учебная практика «Flat Ассемблер»**

*Теоретические сведения:*

**Команды INT и IRET: работа с прерываниями**

Прерыванием называется такое событие, когда процессор приостанавливает нормальное выполнение программы и начинает выполнять другую программу (подпрограмму), предназначенную для обработки прерывания. Закончив обработку прерывания, он возвращается к выполнению приостановленной программы.

*Давайте рассмотрим пример. Я сижу за столом и читаю книгу. С точки зрения компьютера, я выполняю процесс чтения книги. Внезапно звонит телефон — я прерываю чтение, кладу в книгу закладку (на языке процессора это называется «сохранить контекст») и беру трубку. Теперь я «обрабатываю» телефонный звонок. Закончив разговор, я возвращаюсь к чтению книги. Найти последнее прочитанное место помогает та самая закладка.*

Все прерывания делятся на две группы: программные и аппаратные. Программные прерывания порождаются по команде INT. Программные прерывания можно рассматривать как «прерывания по требованию», например, когда вызывается подпрограмма операционной системы для вывода строки символов. В случае с программным прерыванием программист сам определяет, какое прерывание будет вызвано в тот или иной момент.

Команде INT нужно передать всего один 8-битный операнд, который задает номер нужного прерывания.

INT o1

Аппаратные прерывания вызываются аппаратными средствами компьютера и периферийными устройствами, подключенными к общей шине (ISA или PCI). Устройство, запрашивающее прерывание, генерирует так называемый запрос на прерывание (IRQ, interrupt requests). Всего существует 16 аппаратных запросов на прерывание (IRQ0 — IRQ15), поскольку только 16 проводников в шине ISA выделено для этой цели.

К аппаратным прерываниям относятся также специальные прерывания, которые генерирует сам процессор. Такие прерывания используются для обработки «исключительных ситуаций» — неверный операнд, неизвестная команда, переполнение и другие непредвиденные операции, когда процессор сбит с толку и не знает, что делать. Эти прерывания имеют свои обозначения и никак не относятся к зарезервированным для периферии прерываниям.

Все аппаратные прерывания можно разделить на две группы: прерывания, которые можно игнорировать («замаскировать») и те, которые игнорировать нельзя. Первые называются маскируемыми (maskable), а вторые — немаскируемыми (non-maskable). Маскируемые аппаратные прерывания могут быть отключены путем установки флага IF регистра флагов в 0. Примерами немаскируемых прерываний являются прерывания, генерируемые при сбое памяти, сбоях в питании процессора и подобных обстоятельствах.

Запрос на прерывание направляется контроллеру прерываний, который, в свою очередь, передает его микропроцессору. Вместе с запросом процессору передается номер прерывания.

Когда процессор получает номер прерывания, он помещает в стек контекст выполняемой в данный момент программы..

Роль контекста играют значения CS, (Е)IР и регистр флагов.

Теперь процессор будет выполнять другую программу — обработчик прерывания. Адрес этой программы называется вектором прерывания. Векторы прерывания хранятся в таблице векторов прерываний, находящейся в памяти. Таблицу прерываний можно представить себе как массив адресов подпрограмм, в котором индекс массива соответствует номеру прерывания. Процессоры семейства х86 и совместимые с ними могут порождать до 256 прерываний.

После того, как процессор определит адрес обработчика прерывания, он запишет его в пару CS и (Е)IР. Следующая выполненная команда будет первой командой обработчика прерывания.

Возврат из обработчика прерывания осуществляется с помощью команды

IRET, которая восстанавливает исходные значения (E)IP, CS и флагов из стека. Формат команды.

Для того чтобы связать адрес обработчика прерывания с номером прерывания, используется таблица векторов прерываний, занимающая первый килобайт оперативной памяти - адреса от 0000:0000 до 0000:03FF. Таблица состоит из 256 элементов - FAR-адресов обработчиков прерываний. В первом слове элемента таблицы записано смещение, а во втором - адрес сегмента обработчика прерывания.

Прерыванию с номером 0 соответствует адрес 0000:0000, прерыванию с номером 1 - 0000:0004 и т.д. Инициализация таблицы происходит частично BIOS после тестирования аппаратуры и перед началом загрузки операционной системой, частично при загрузке ОС. ОС может переключить на себя некоторые прерывания BIOS. Затем значения большинства векторов возможно изменить в процессе выполнения пользовательских программ.

Приведем назначение некоторых наиболее важных векторов:

|  |  |
| --- | --- |
| Номер | Описание |
| 0 | Ошибка деления. Вызывается автоматически после выполнения команд DIV или IDIV, если в результате деления происходит переполнение (например, при делении на 0). DOS обычно при обработке этого прерывания выводит сообщение об ошибке и останавливает выполнение программы. |
| 1 | Прерывание пошагового режима. Вырабатывается после выполнения каждой машинной команды, если в слове флагов установлен бит пошаговой трассировки TF. Используется для отладки программ. |
| 2 | Аппаратное немаскируемое прерывание. Это прерывание может использоваться по-разному в разных машинах. |
| 3 | Прерывание для трассировки. Это прерывание генерируется при выполнении однобайтовой машинной команды с кодом CCh и обычно используется отладчиками для установки точки прерывания. |
| 4 | Переполнение. Генерируется машинной командой INTO, если установлен флаг OF. Если флаг не установлен, то команда INTO игнорируется. Это прерывание используется для обработки ошибок при выполнении арифметических операций. |
| 5 | Печать копии экрана. Генерируется при нажатии на клавиатуре клавиши PrtScr. |
| 6 | Неопределенный код операции или длина команды больше 10 байт (для процессора 80286). |
| 7 | Особый случай отсутствия математического сопроцессора (процессор 80286). |
| 8 | IRQ0 - прерывание интервального таймера, возникает 18,2 раза в секунду. |
| 9 | IRQ1 - прерывание от клавиатуры. Генерируется при нажатии и при отжатии клавиши. Используется для чтения данных от клавиатуры. |
| B | IRQ3 - прерывание асинхронного порта COM2. |
| C | IRQ4 - прерывание асинхронного порта COM1. |
| D | IRQ5 - прерывание от контроллера жесткого диска для. |
| E | IRQ6 - прерывание генерируется контроллером флоппи-диска после завершения операции. |
| F | IRQ7 - прерывание принтера. Генерируется принтером, когда он готов к выполнению очередной операции. |
| 10 | Обслуживание видеоадаптера. |
| 11 | Определение конфигурации устройств в системе. |
| 12 | Определение размера оперативной памяти в системе. |
| 13 | Обслуживание дисковой системы. |
| 14 | Последовательный ввод/вывод. |
| 16 | Обслуживание клавиатуры. |
| 17 | Обслуживание принтера. |
| 19 | Загрузка операционной системы. |
| 1A | Обслуживание часов. |
| 1B | Обработчик прерывания Ctrl-Break. |
| 20-5F | Используется DOS или зарезервировано для DOS. |
| 60-67 | Прерывания, зарезервированные для пользователя. |
| 75 | IRQ13 - прерывание от математического сопроцессора. |
| 76 | IRQ14 - прерывание от контроллера жесткого диска. |

Важные прерывания DOS:

INT 20H. Завершение программы.

Запрос завершает выполнение программы и передает управление в DOS. Данный запрос обычно находится в основной процедуре.

INT 21H. Запрос функций DOS.

Основная операция DOS, вызывающая определенную функцию в соответствии с кодом в регистре AH. Назначение функций DOS описано далее.

INT 22H. Адрес подпрограммы обработки завершения задачи.

INT 23H. Адрес подпрограммы реакции на Ctrl/Break..

INT 24H. Адрес подпрограммы реакции на фатальную ошибку.

В этом элементе и в двух предыдущих содержатся адреса, которые инициализируются системой в начале программного сегмента и, которые можно изменить для своих целей.

INT 25H. Абсолютное чтение с диска.

INT 26H. Абсолютная запись на диск

INT 27H.Завершение программы, оставляющее ее резедентной.

Позволяет сохранить COM-программу в памяти.

Ниже приведены базовые функции для прерывания DOS INT 21H. Код функции устанавливается в регистре AH:

00 Завершение программы (аналогично INT 20H).

01 Ввод символа с клавиатуры с эхом на экран.

02 Вывод символа на экран.

03 Ввод символа из асинхронного коммуникационного канала.

04 Вывод символа на асинхронный коммуникационный канал.

05 Вывод символа на печать .

06 Прямой ввод с клавиатуры и вывод на экран.

07 Ввод с клавиатуры без эха и без проверки Ctrl/Break.

08 Ввод с клавиатуры без эха с проверкой Ctrl/Break.

09 Вывод строки символов на экран.

0А Ввод с клавиатуры с буферизацией.

0В Проверка наличия ввода с клавиатуры.

0С Очистка буфера ввода с клавиатуры и запрос на ввод.

0D Сброс диска.

0Е Установка текущего дисковода.

0F Открытие файла через.

10 Закрытие файла через.

11 Начальный поиск файла по шаблону.

12 Поиск следующего файла по шаблону.

13 Удаление файла с диска.

14 Последовательное чтение файла.

15 Последовательная запись файла.

16 Создание файла.

2А Получение даты (CX-год,DН-месяц,DL-день).

2В Установка даты.

2С Получение времени (CH-час,CL-мин,DН-с,DL-1/100с).

2D Установка времени.

4В Загрузка/выполнение программы (подпроцесса).

4С Завершение подпроцесса с возвратом управления.

4D Получение кода завершения подпроцесса.

Обычно параметр функции передается через регистры DS:DX, а возвращаемое значение через регистр АL.

Таким образом, чтобы например вывести строку символов на экран через консоль DOS, необходимо выполнить код:

mov ah, 09h ;выбор нужной функции (вывод строки)

mov dx, str\_adr ;загрузка адреса подготовленной строки символов

int 21h ;вызов прерывания функции DOS

**Псевдокоманды**

Операторы языка ассемблера делятся на команды и псевдокоманды (директивы).

Команды ассемблера — это символические имена машинных команд, обработка их компилятором приводит к генерации машинного кода. Псевдокоманды же управляют работой самого компилятора. В машинный код они не преобразуются, а просто выполняют какие-либо действия в процессе компиляции (аналогично меткам). На одной и той же аппаратной архитектуре могут работать различные ассемблеры: их команды обязательно будут одинаковыми, но псевдокоманды могут быть разными. В качестве примера рассмотрим псевдокоманды компилятора FASMW для языка. Flat Assembler.

**Псевдокоманды DB, DW и DD — определение констант**

Чаще всего используется псевдокоманда DB (define byte), позволяющая определить числовые данные и строки, каждый элемент которых будет имеет размерность в 1 байт. Рассмотрим несколько примеров:

db 0x55 ;один байт 0х55 в шестнадцатеричном виде

db 0x55,0x56,0x57 ;три последовательных байта: 0x55, 0х56, 0x57

db 'а',0x55 ;можно записать символ в одинарных кавычках. Получится

;последовательность 0x61, 0x55

db 'Hello',13,10,'$' ;можно записать целую строку. Получится 0x48, 0x65, Ох6С,

;Ох6С, 0x6F, OxD, OxA, 0x24

Символ доллара в кавычках означает конец строки, а символ доллара без кавычек – адрес текущей команды.

Для определения порции данных размера, кратного слову, служит директива DW (define word):

dw 0x1234 ;0х34, 0x12

dw ' a ' ;0x61, 0x00: второй байт заполняется нулями

Директива DD (define double word) задает значение порции данных размера, кратного двойному слову:

dd 0x12345678 ;0х78 0x56 0x34 0x12

Чтобы связать константу с определенным буквенным обозначением, необходимо сделать следующее:

number dd 0x1 ;идентификатор number инициализирован значением 1

«number» теперь представляет адрес в памяти, по которому записано значение 0x00000001 длиной в двойное слово.

**Псевдокоманды RESB, RESWH RESD — объявление переменных**

Вторая группа псевдокоманд позволяет определить неинициализированные данные. Если в первом случае мы заносим данные в память, то сейчас мы просто укажем, сколько байтов нужно зарезервировать для будущих данных.

Память для этих неинициализированных переменных распределяется динамически, а сами данные не содержатся в исполнимом файле.

Для резервирования памяти служат три директивы: RESB (резервирует байт), RESW (резервирует слово) и RESD (резервирует двойное слово). Аргументом этих псевдокоманд является количество резервируемых позиций:

number resd l ;резервирует 4 байта для переменной "number"

buffer resb 64 ;резервирует 64 байта для ;переменной buffer

**Псевдокоманда DUP - повторение псевдокоманды**

Директива DUP — это псевдокоманда постфиксного типа, то есть она используется только после другой команды. Она повторяет предыдущую псевдокоманду указанное количество раз Применяется эта директива в тех случаях, когда нужно «забить» некоторую область памяти повторяющимся образцом.

Синтаксис команды следующий:

o1 o2 dup (o3)

Где о1 – псевдокоманда, которую нужно повторять, о2 – количество повторений, о3 – параметр повторяющейся операции. Например, создадим массив из 5 нулевых элементов:

mass db 5 dup (0)

Если параметр о3 неизвестен – необходимо на его месте ставит знак ?.

**Псевдокоманда EQU — вычисление истинных константных выражений**

Эта директива определяет константу, известную во время компиляции. В качестве значения константы можно указывать также константное выражение.

Директиве EQU должно предшествовать символическое имя:

four EQU 4 ;тривиальный пример.

**Подключение файлов с кодом**

«include» включает указанный файл с исходным кодом туда, где эта директива используется. За ней должно следовать в кавычках имя файла, который должен быть включен, например:

include 'macros.inc'

Весь включенный файл обрабатывается компилятором перед обработкой строк, следующих за содержащей директиву «include». Нет предела для количества включаемых файлов, пока они умещаются в память.

**Работа с адресами**

«org» устанавливает адрес в памяти, по которому загружается следующий за ней код. В качестве операнда используется числовое выражение, указывающее адрес.

«load» позволяет определить константу двоичным значением, загруженным из уже сассемблированного кода. За директивой должно следовать имя константы, затем необязательный оператор размера, затем оператор «from» и числовое выражение, определяющее адрес в текущем адресном пространстве.

«store» может модифицировать уже сгенерированный код заменой некоторых ранее сгенерированных байтов значением, задаваемым следующим за инструкцией числовым выражением. Перед этим выражением может идти оператор размера, определяющий, насколько длинное значение оно задает, то есть сколько будет сохранено байт. Далее должен следовать оператор «at» и числовое выражение, указывающее адрес в текущем адресном пространстве кода. По этому адресу будет сохранено задаваемое значение.

«virtual» определяет виртуальные данные по указанному адресу. Эти данные не будут включены в файл вывода, но метки, определенные здесь, могут использоваться в других частях кода. За этой директивой может следовать оператор «at» и числовое выражение, определяющее адрес виртуальных данных, иначе будет использован текущий адрес. Инструкции определяемых данных должны быть расположены на следующих строках и заканчиваться директивой «end virtual».

**Выбор разрядности генерируемого кода:**

«use16» и «use32» указывают ассемблеру генерировать 16-битный или 32-битный код, пренебрегая настройкой по умолчанию для выбранного формата вывода. «use64» включает генерирование кода для длинного режима процессоров x86.

**Смена формата выходного файла.**

Выполняется с помощью директивы format. данная директива должна находится в самом начале программного кода.

В качестве параметра можно указать следующие форматы выходных файлов:

1) MZ — исполняемые файлы DOS.

При использовании данного формата возможно применение следующих директив:

«segment» определяет новый сегмент, за ним должна следовать метка, чьим значением будет номер определяемого сегмента. Все метки, определенные далее, будут иметь значения относительно начала этого сегмента.

«entry» устанавливает точку входа в программу, за ней должен следовать дальний адрес (имя сегмента, двоеточие и смещение в сегменте) желаемой точки входа.

«stack» устанавливает стек. За директивой может следовать числовое выражение, указывающее размер стека для автоматического создания, либо дальний адрес начала стека если вы хотите установить стек вручную.

«heap» со следующим за ней значением определяет максимальный размер дополнительного места в памяти. Чтобы всегда отводить только память, которая программе действительно нужна лучше использовать «heap 0».

2) PE — исполняемые файлы Windows (консольные «console», графические приложения «GUI», и динамические библиотеки «DLL»).

При использовании данного формата возможно применение следующих директив:

«section» определяет новую секцию, за ней должна следовать строка в кавычках, определяющая имя секции, и далее могут следовать один или больше флагов секций. Возможные флаги такие: «code», «data», «readable», «writeable», «executable», «shareable», «discardable», «notpageable». Пример объявления секции PE:

section '.text' code readable executable

Вместе с флагами также может быть определен один из специальных идентификаторов данных PE, отмечающий всю секцию как специальные данные, возможные идентификаторы: «export», «import», «resource» и «fixups».

«data» начинает определение специальных данных PE, за директивой должен следовать один из идентификаторов данных («export», «import», «resource» или «fixups») или номер записи данных в заголовке PE. Данные должны быть определены на следующих строках и заканчиваться директивой «end data».

Также применяются «entry», «stack», «heap».

3) PE64 — исполняемые файлы 64-битных версий Windows.

4) COFF, MS COFF, MS64 COFF — объектные файлы.

5) ELF, ELF64 — исполняемые файлы в UNIX-подобных системах.

6) Binary — файлы произвольной структуры.

*Пример программы на «чистом» Ассемблере для вывода строки «Hello World»:*

format MZ ;Определяем тип выходного файла (ехе)

org 100h ;Переходим на адрес 100h сегмента кода программы

mov ah, 09h ;Загружаем номер вызываемой прерыванием функции

;(09 - вывод строки символов на экран)

mov dx, c ;В качестве параметра передаем адрес, хранящий начало строки

int 21h ;Вызываем программное прерывание DOS

mov ah, 01h ;Реализуем задержку отображения путем применения функции

int 21h ;ввода символа

mov ah, 4ch ;Завершаем процесс программы с передачей управления

mov al, 0h ;На всякий случай обнуляем al перед завершением

int 21h

c db "Hello world$" ;Объявляем и инициализируем набор битовых констант для

;хранения строки символов

*Пример программы на «чистом» Ассемблере для вывода восьмибитного целого беззнакового числа в строковом формате:*

format MZ

org 100h

;Для вывода числа на экран, необходимо его разделить на

;отдельные цифры, преобразовать каждую цифру в символ, и

;собрать символы в строку

mov bl, 10 ;Инициализируем делитель

mov dl, [x] ;Загружаем значение выводимой переменной

mov dh, 0

mov di, str1 ;Загружаем адрес первого символа выводимой строки

;в требуемый индексный регистр

add di, 4 ;Увеличиваем адрес на 4 байта, чтобы не изменить

;вспомогательную строку "a = "

;Последовательность символов будем сохранять в стэк

mov al, '$' ;Первым элементом (который будет выводиться последним)

mov ah, 0 ;загружаем символ конца строки

push ax

label1:

mov ax, dx ;Делим число на 10

div bl

mov dl, al ;Целую часть возвращаем в исходный регистр dx для

mov dh, 0 ;следующего «витка» цикла обработки

mov al, ah ;Остаток является обрабатываемой цифрой, грузим его в al

add al, 0x30 ;Преобразуем цифру в символ, путем добавления кода,

;соответствующего символу '0' (см. таблицу ASCII)

mov ah, 0

push ax ;Загружаем символ цифры в стэк

cmp dl, 0 ;Проверяем конец числа

jne label1 ;Переходим на новый «виток»

;Формируем строку из набора символов

label2:

pop ax ;Извлекаем символ из стэка

stosb ;Загружаем символ в память, выделенную для строки

cmp al, '$' ;Проверяем последний символ строки

jne label2

mov ah,09h

mov dx,str1

int 21h

mov ah,01h

int 21h

mov ah, 4ch

mov al, 0h

int 21h

str1 db "a = " ;Резервируем и частично инициализируем память под строку

x db 55 ;Выделяем память и задаем восьмибитное число

*Задание:* Разработать программу на «чистом» Ассемблере (FASM) в соответствии с заданием.

*Условия:* В программе обязательно реализуется ввод и вывод целых чисел через консоль DOS на чистом Ассемблере. В программе должны обязательно использоваться подпрограммы и стэк.

|  |  |  |
| --- | --- | --- |
| **№** | **Условие задачи** | **ФИО студентов** |
| 1 | Даны натуральное число n, целые числа a1,...,an. Получить удвоенную сумму всех положительных членов последовательности a1,...,an. | 1.  2. |
| 2 | Даны натуральное число n, целые числа a1,..., an. В последовательности a1,...,an все отрицательные члены увеличить на 5, а все неотрицательные заменить на 1. | 1.  2. |
| 3 | Даны целые числа a0, ..., a6. Получить для x = 1, 3, 4 значения p(x + 1) - p(x), где p(y) = a6y6 + a5y5 + ... + a0. | 1.  2. |
| 4 | Даны целые числа a1, ..., an; b1, ..., bm. В последовательности a1, ..., an и в последовательности b1, ..., bm все члены, следующие за членом с наибольшим значением (за первым по порядку, если их несколько), заменить на 1. | 1.  2. |
| 5 | Даны целые числа a0, ..., an; b1, ..., bm; k. Если в последовательности a0, ..., an нет ни одного члена со значением k , то первый по порядку член этой последовательности, не меньший всех остальных членов, заменить на значение k. По такому же правилу преобразовать последовательность b1, ..., bm применительно к значению 10. | 1.  2. |
| 6 | Дано натуральное число n, целые числа a1, ..., an. Рассмотреть отрезки последовательности a1, ..., an (подпоследовательности идущих подряд членов), состоящие из полных квадратов. Получить наибольшую из длин рассматриваемых отрезков. | 1.  2. |
| 7 | Дано натуральное число n, целые числа a1, ..., an. Рассмотреть отрезки последовательности a1, ..., an (подпоследовательности идущих подряд членов), состоящие из степеней пятерки. Получить наибольшую из длин рассматриваемых отрезков. | 1.  2. |
| 8 | Дано натуральное число n, целые числа a1, ..., an. Рассмотреть отрезки последовательности a1, ..., an (подпоследовательности идущих подряд членов), состоящие из простых чисел. Получить наибольшую из длин рассматриваемых отрезков. | 1.  2. |
| 9 | Даны целые числа x1, y1, x2, y2, ..., x10, y10. Найти периметр десятиугольника, вершины которого имеют соответственно координаты (x1, y1), ( x2, y2), ..., (x10, y10). | 1.  2. |
| 10 | Даны натуральные числа n, a1...an. Определить количество членов ak последовательности a1,...,an являющихся нечетными числами. | 1.  2. |
| 11 | Даны натуральные числа n, a1...an. Определить количество членов ak последовательности a1,...,an являющихся квадратами четных чисел. | 1.  2. |
| 12 | Даны натуральные числа n, a1...an. Определить количество членов ak последовательности a1,...,an кратных 3 и не кратных 5. | 1.  2. |
| 13 | Даны натуральные числа n, a1...an. Определить количество членов ak последовательности a1,...,an удовлетворяющих условию 2k < ak < k!. | 1.  2. |
| 14 | Даны натуральные числа n, a1...an. Определить количество членов ak последовательности a1,...,an имеющих четные порядковые номера и являющихся нечтными числами. | 1.  2. |
| 15 | Данны натуральные числа n, q1,...,qn. найти те члены qi последовательности q1,...,qn, которые при делении на 7 дают остаток 1, 2 или 5. | 1.  2. |
| 16 | Данны натуральные числа n, q1,...,qn. найти те члены qi последовательности q1,...,qn, которые являются удвоенными нечетными числами. | 1.  2. |
| 17 | Даны целые числа a1,...a10. Получить сумму тех чисел данной последовательности, которые кратные 5. | 1.  2. |
| 18 | Даны целые числа a1,...a10. Получить сумму тех чисел данной последовательности, которые нечетные и отрицательные. | 1.  2. |
| 19 | Даны целые числа a1,...a15. Получить сумму тех чисел данной последовательности, которые удовлетворяют условию |ai| < i2 | 1.  2. |
| 20 | Даны натуральные числа n, p, целые числа a1,...,an. Получить произведение членов последовательности a1,...,an, кратных p. | 1.  2. |
| 21 | Даны натуральные числа n, a1...an. Определить количество членов ak последовательности a1,...,an, являющихся четными числами. | 1.  2. |
| 22 | Даны натуральные числа n, a1,..., an. В последовательности a1,...,an все отрицательные члены увеличить на 5, а все неотрицательные заменить на 1. | 1.  2. |
| 23 | Даны натуральные числа n, a1,...,an. Получить все натуральные j (2 ≤ j ≤ n-1), для которых aj-1 < aj > aj+1. | 1.  2. |
| 24 | Даны натуральные числа n, a1,...,an. В последовательности a1,...,an, определить число соседств двух чисел разного знака. | 1.  2. |
| 25 | Дано натуральное число n. Получить все его натуральные делители | 1.  2. |