Министерство образования Российской Федерации

Волжский университет им. Татищева

Факультет «Информатика и телекоммуникации»

Кафедра «Информатика и системы управления»

УТВЕРЖДАЮ

Проректор по учебной работе

\_\_\_\_\_\_\_\_\_\_\_\_\_Е.В. Филатова

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 200 г.

**МЕТОДИЧЕСКОЕ УКАЗАНИЕ**

для проведения лабораторной работы по теме

''Программирование контроллера приоритетных прерываний''

по курсу ''Организация ЭВМ''

для студентов специальностей 220100, 071900

Тольятти

**Создание и компиляция программ на ассемблере**

Процесс разработки программы на ассемблере состоит из пяти этапов:

1. Создание файла с исходным текстом программы в любом текстовом редакторе. Расширение файла с исходным текстом может быть .asm, или .txt, или .doc.
2. Создание объектного модуля. В среде DOS или NORTON или FAR в командной строке набираете следующую команду:

**tasm name.asm**

или

**tasm.exe name.asm name.obj**

name.asm файл с исходным текстом программы. При этом файлы tasm.exe и name.asm должны находится в одном каталоге. После запуска этой команды мы получаем объектный файл с расширением .obj. Если объектный файл не появился, то в программе содержатся ошибки. Перечень ошибок можно посмотреть, отключив панели (ctrl+o или Ctrl+f1 и ctrl+f2).

1. Создание исполнительного файла. В командной строке набираем следующую команду:

**tlink name.obj**

или

**tlink.exe name.obj name exe**

При этом файлы tlink.exe и name.obj должны находится в одном каталоге. После запуска этой команды мы получаем запускной файл с расширением .exe. Если запускной файл не появился в этом каталоге, то в данном каталоге не хватает некоторых библиотек. Перечень файлов можно посмотреть, отключив панели (ctrl+o или Ctrl+f1 и ctrl+f2).

1. Тестирование программы. Запустите исполнительный файл.
2. Пошаговая отладка. В командной строке набираем следующую команду:

**td name.exe**

**Структура программы на ассемблере**

**Model small** ;модель программы, или же количество памяти на сегмент

**.data** ;сегмент данных

;описание переменных

**.stack 100h** ;сегмент стека

**.code ;**сегмент данных

;процедуры, макрокоманды

**main:**

**mov ax,@data**

**mov ds,ax**

;основная программа

**mov ax,4c00h**

**int 21h** ;выход из программы

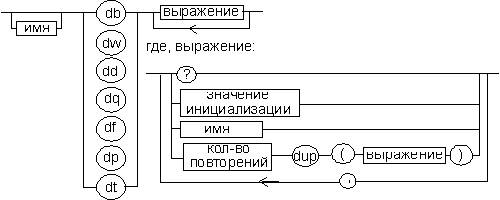
**end main**

**Директивы резервирования памяти**

Для описания простых типов данных в программе используются специальные директивы резервирования и инициализации данных, которые, по сути, являются указаниями транслятору на выделение определенного объема памяти. Если проводить аналогию с языками высокого уровня, то директивы резервирования и инициализации данных являются определениями переменных.

**Машинного эквивалента этим директивам нет; просто транслятор, обрабатывая каждую такую директиву, выделяет необходимое количество байт памяти и при необходимости инициализирует эту область некоторым значением.**

Директивы резервирования и инициализации данных простых типов имеют формат:



**Рис. 1.** Директивы описания данных простых типов

На рис. 1 использованы следующие обозначения:

* **?** показывает, что содержимое поля не определено, то есть при задании директивы с таким значением выражения содержимое выделенного участка физической памяти изменяться не будет. Фактически, создается неинициализированная переменная;
* **значение инициализации** — значение элемента данных, которое будет занесено в память после загрузки программы. Фактически, создается инициализированная переменная, в качестве которой могут выступать константы, строки символов, константные и адресные выражения в зависимости от типа данных. Подробная информация приведена в приложении 1;
* **выражение** — итеративная конструкция с синтаксисом, описанным на рис. 5.17. Эта конструкция позволяет повторить последовательное занесение в физическую память выражения в скобках n раз.
* **имя** — некоторое символическое имя метки или ячейки памяти в сегменте данных, используемое в программе.
* **db** **— резервирование памяти для данных размером 1 байт.** Директивой **db** можно задавать следующие значения:
  + выражение или константу, принимающую значение из диапазона:

для чисел со знаком –128...+127; для чисел без знака 0...255;

* + символьную строку из одного или более символов. Строка заключается в кавычки. В этом случае определяется столько байт, сколько символов в строке.
* **dw — резервирование памяти для данных размером 2 байта**.
  + выражение или константу, принимающую значение из диапазона:

для чисел со знаком –32 768...32 767; для чисел без знака 0...65 535;

* + выражение, занимающее 16 или менее бит, в качестве которого может выступать смещение в 16-битовом сегменте или адрес сегмента;
  + 1- или 2-байтовую строку, заключенная в кавычки.
* **dd** — **резервирование памяти для данных размером 4 байта.**
  + выражение или константу, принимающую значение из диапазона:

для чисел со знаком –2 147 483 648...+2 147 483 647;

для чисел без знака 0...4 294 967 295;

* + относительное или адресное выражение, состоящее из 16-битового адреса сегмента и 16-битового смещения;
  + строку длиной до 4 символов, заключенную в кавычки.
* **df — резервирование памяти для данных размером 6 байт;**
* **dp** — **резервирование памяти для данных размером 6 байт.** Директивами **df** и **dp** можно задавать следующие значения:
  + выражение или константу, принимающую значение из диапазона:

для чисел со знаком –2 147 483 648...+2 147 483 647;

для чисел без знака 0...4 294 967 295;

* + относительное или адресное выражение, состоящее из 32 или менее бит (для i80386) или 16 или менее бит (для младших моделей микропроцессоров Intel);
  + адресное выражение, состоящее из 16-битового сегмента и 32-битового смещения;
  + строку длиной до 6 байт, заключенную в кавычки.
* **dq — резервирование памяти для данных размером 8 байт.**

относительное или адресное выражение, состоящее из 32 или менее бит

* + константу со знаком из диапазона –263...263–1;
  + константу без знака из диапазона 0...264–1;
  + строку длиной до 8 байт, заключенную в кавычки.
* **dt — резервирование памяти для данных размером 10 байт.**

относительное или адресное выражение, состоящее из 32 или менее бит

* + адресное выражение, состоящее из 16-битового сегмента и 32-битового смещения;
  + константу со знаком из диапазона –279...279-1;
  + константу без знака из диапазона 0...280-1;
  + строку длиной до 10 байт, заключенную в кавычки;
  + упакованную десятичную константу в диапазоне 0...99 999 999 999 999 999 999.

Очень важно уяснить себе порядок размещения данных в памяти. Он напрямую связан с логикой работы микропроцессора с данными. Микропроцессоры Intel требуют следования данных в памяти по принципу: **младший байт по младшему адресу**.

Для иллюстрации данного принципа рассмотрим пример 1, в котором определим сегмент данных. В этом сегменте данных приведено несколько директив описания простых типов данных.

Пример 1. Пример использования директив резервирования и инициализации данных. Результатом работы данной программы будет строка 'Привет, все работает'

**model small**

**.stack 100h**

**.data**

mes db 'Привет, все работает',’$' ;определение строки

perem\_1 db 0ffh ;определение контстанты

perem\_2 dw 3a7fh ;определение контстанты

perem\_3 dd 0f54d567ah ;определение контстанты

mas db 10 dup (' ') ;определение пустого массива из 10 байт

adr dw perem\_3 ;переменная adr содержит адрес

;внутри сегмента переменной perem\_3

a\_full dd perem\_3 ;переменная a\_full содержит полный

;адрес переменной perem\_3

fin db 'Конец сегмента данных программы $'

**.code**

**start:**

;занесение в сегментный регистр адреса сегмента данных

**mov ax,@data**

**mov ds,ax**

mov ah,09h

mov dx,offset mes

int 21h ;вывод на экран строки mes

**mov ax,4c00h**

**int 21h** ;выход из программы

**end start**

Окончание работы программы сопровождается полной выгрузкой программы из оперативной памяти, это осуществляется функцией 4с00h прерывания int 21h.

Все что в данной программе выделено жирным шрифтом обязательно при написании любой программы.

При написании программ на ассемблере регистр букв не важен.

## Организация ввода-вывода на ассемблере

Ввод-вывод данных в компьютер осуществляется посредством различных периферийных устройств. Общение процессора с различными периферийными происходит через систему прерываний. Для ввода-вывода данных служит прерывание int 21h.

Основная последовательность дей­ствий при использовании прерываний 2lh (DOS):

1. Поместить номер функции в регистр ah.

2. Поместить передаваемые функции параметры в определенные регистры (они приведены при описании каждой функции).

3. Вызвать прерывание 2lh (DOS) командой int 21h

4. Извлечь результаты работы функций из определенных регистров. Какие именно регистры и что они содержат после возврата управления из функ­ции программе пользователя, указывается при описании каждой функции.

Прерывание DOS 2lh предназначено для предоставления программисту раз­личных услуг со стороны операционной системы. Этими услугами является набор функций. Какая именно функция должна быть вызвана, указывается числом в регистре ah.

**Некоторые функции DOS (int 21h)**

|  |  |  |  |
| --- | --- | --- | --- |
| Назначение | Номер функции | Вход | Выход |
| Ввод символа с ожиданием и эхосопровождением | ah-0lh |  | аl- ASCII-код символа |
| Вывод символа | ah-02h | dl- ASCII-код символа |  |
| Вывод символа на принтер | ah-05h | dl- ASCII-код символа |  |
| Ввод символа с ожиданием и без эхосопровождения | ah-07h  ah-08h |  | al- ASCII-код символа (функция 08h при вводе проверяет, не нажато ли CTRL-BREAK) |
| Вывод строки на экран | ah-09h | ds:dx = адрес строки с символом <$> на конце | Введенная строка в буфере |
| Ввод строки с клавиатуры | ah-0ah | ds:dx-anpec буфера с форматом:  1 байт — размер буфера для ввода (формирует пользователь);  2 байт — число фактически введенных символов (заполняет система по окончанию ввода — нажатию клавиши Enter (Odh)). Символ 0dh не учитывается во втором байте буфера;  3 байт и далее — введенная строка с символом 0dh на конце |  |
| Проверка состояния буфера клавиатуры | ah - 0bh |  | al=0 — буфер пуст  al = 0ffh - в буфере есть символы |

Пример 2. Программа ввода символа с клавиатуры

**model small**

**.stack 100h**

**.data**

**.code**

**start:** ;занесение в сегментный регистр адреса сегмента данных

**mov ax, @data**

**mov ds, ax**

;помещаем в регистр ah номер функции, которая вводит символ

mov ah, 01h

int 21h ;вводим символ с клавиатуры

;символ введенный с клавиатуры находится в регистре al

**mov ax,4c00h**

**int 21h** ;выход из программы

**end start**

Пример 3. Программа вывода символа на экран

**model small**

**.stack 100h**

**.data**

f db 'ф' ;помещаем в переменную f выводимый символ

**.code**

**start:** ;занесение в сегментный регистр адреса сегмента данных

**mov ax, @data**

**mov ds, ax**

;помещаем в регистр ah номер функции, которая выводит символ

mov ah, 02h

mov dl, f ; помещаем в dl символ

int 21h ;выводим символ на экран

**mov ax,4c00h**

**int 21h** ;выход из программы

**end start**

Пример 4. Вывод строки на экран

**model small**

**.stack 100h**

**.data**

f db 'строка вывода$'

;f – строковая переменная, которая обязательно заканчивается знаком $

**.code**

**start:** ;занесение в сегментный регистр адреса сегмента данных

**mov ax, @data**

**mov ds, ax**

;помещаем в регистр ah номер функции, которая выводит строку на экран

mov ah, 09h

mov dx, offset f ; помещаем в dx адрес строки, которую выводим

int 21h ;выводим строку

**mov ax,4c00h**

**int 21h** ;выход из программы

**end start**

**Организация вычислений**

## Логические команды

Система команд микропроцессора содержит пять логических команд. Эти команды выполняют логические операции над битами операндов. Размерность операндов должна быть одинакова. В качестве операндов могут использоваться, регистры, ячейки памяти (переменные) и непосредственные операнды (числа). Любая логическая команда меняет значение следующих флагов of, sf,zf,pf,cf (переполнение, знак, нуля, паритет, перенос)

[***and***](D:DASHKAПреподавательствоОрганизацияЭВМDASHKALibrari%20ot%20DariaAssemblerAssemblerguideTextCommand.htm#and) **операнд\_1,операнд\_2** — операция логического умножения (И - конъюнкция).

and ah, 0a1h; ah:=ah٧0ah

and bx, cx; bx:=bx٧cx

and dx, x1; dx:=dx٧x1

Команда and может применяться для сброса определенных битов в 0 или для определения значения некоторых битов.

Например, необходимо 5й бит числа находящегося в bl установить в 0, остальные биты не трогать.

and bl, 11011111b или and bl, 0cfh

Если необходимо определить чему равен 5й бит, то

and bl, 00100000b или and bl, 20h

В результате если в регистре bl в 5м бите был 0, то после выполнения этой команды мы получим нулевой результат, обнулим весь регистр. Если же в регистре bl в 5м бите была 1, то мы получим не нулевой результат.

[***or***](D:DASHKAПреподавательствоОрганизацияЭВМDASHKALibrari%20ot%20DariaAssemblerAssemblerguideTextCommand.htm#or) **операнд\_1,операнд\_2** — операция логического сложения (ИЛИ - дизъюнкцию)

or al, x1; al:=al & x1

or eax,edx; eax:=eax & edx

or dx, 0fa11h; dx:=dx & 0fa11h

Команда or может применяться для установки определенных бит в 1. Например, необходимо установить в единицу 4й и 7й биты регистра ah.

or ah, 10010000b или or ah, 90h

[***xor***](D:DASHKAПреподавательствоОрганизацияЭВМDASHKALibrari%20ot%20DariaAssemblerAssemblerguideTextCommand.htm#xor) **операнд\_1,операнд\_2** — операция логического исключающего сложения (исключающего ИЛИ ИЛИ-НЕ). Команда может применятся для выяснения того какие биты в операндах различаются ил для инвертирования состояния заданных бит в операнде\_1. Например, необходимо определить совпадает ли содержимое регистров ax и dx

xor ax, dx ;если содержимое совпадает то в регистре ах мы получим

;нулевой результат, иначе не нулевой результат.

xor bh,10b ; инвертировали 1й бит в регистре bh

[***test***](D:DASHKAПреподавательствоОрганизацияЭВМDASHKALibrari%20ot%20DariaAssemblerAssemblerguideTextCommand.htm#test) **операнд\_1,операнд\_2** — операция “проверить” (способом логического умножения). Команда выполняет поразрядно логическую операцию И над битами операндов операнд\_1 и операнд\_2. Состояние операндов остается прежним, изменяются только флаги zf, sf, и pf, что дает возможность анализировать состояние отдельных битов операнда без изменения их состояния.

[***not***](D:DASHKAПреподавательствоОрганизацияЭВМDASHKALibrari%20ot%20DariaAssemblerAssemblerguideTextCommand.htm#not) **операнд** — операция логического отрицания. Команда выполняет поразрядное инвертирование (замену значения на обратное) каждого бита операнда. Результат записывается на место операнда.

not ax ;ax:=

Пример 5. Логическое сложение двух однобайтных чисел.

**model small**

**.stack 100h**

**.data**

x1 db 0c2h ;первое слагаемое

x2 db 022h ;второе слагаемое

y db ? ;переменная результата

**.code**

**start:**

**mov ax,@data**

**mov ds,ax**

mov al, x1 ;в al помещаем первое слагаемое

or al, x2 ;осуществляем логическое сложение, результат в al

mov y, al ;помещаем результат на место

**mov ax,4c00h**

**int 21h**

**end start**

Следующие две команды позволяют осуществить поиск первого установленного в 1 бита операнда. Они появились в 486 процессоре.

**bsf** **операнд\_1, операнд\_2** – сканирование бит операнда\_2 от младшего к старшему в поисках первого бита установленного в 1. Если такой обнаружится, то в операнд\_1 заноситься номер этого бита в целочисленном формате.

Пример:

mov al,02h

bsf bx,al ; bx:=1, т.к. 1й бит регистра al=1

**bsr** **операнд\_1, операнд\_2** – сканирование бит операнда\_2 от старшего к младшему в поисках первого бита установленного в 1. Если такой обнаружится, то в операнд\_1 заноситься номер этого бита в целочисленном формате.

Пример:

mov al,82h

bsr bx,al ; bx:=6, т.к. 6й бит регистра al=1

Если операнд\_2 равен 0 то вышеописанные две команды устанавливают флаг нуля zf в 1, иначе в 0.

## Арифметические операции над целыми двоичными числами

###### Сложение двоичных чисел

***inc* *операнд*** - операция инкремента, то есть увеличения значения операнда на 1;

inc ax; ax:=ax+1

inc x1; х1:=х1+1

***add* *оп1,оп2*** - команда сложения с принципом действия: оп1 = оп1 + оп2 (addition)

add al, bl

add ax, 0fe2h

add ebx, x1+2

add x1, 0fh

add x2, ax

***adc* *оп1,оп2*** - команда сложения с учетом флага переноса cf. оп1 = оп1 + оп2 + знач\_cf

*Вычитание двоичных чисел*

***dec* *операнд*** — операция декремента, то есть уменьшения значения операнда на 1;

dec cx ;cx:=cx-1

dec x

***sub* *операнд\_1,операнд\_2*** — команда вычитания; ее принцип действия:

операнд\_1 = операнд\_1 – операнд\_2

sub al, bl; al:=al-bl

sub ax, x1

sub x2, dx

sub eax, 0f35h

sub x2, 22h

***sbb* *операнд\_1,операнд\_2*** — команда вычитания с учетом заема (флага cf ):

операнд\_1 = операнд\_1 – операнд\_2 – значение\_cf

Пример 6. Сложение двух однобайтных чисел.

**model small**

**.stack 100h**

**.data**

x1 db 0c2h ;первое слагаемое

x2 db 022h ;второе слагаемое

y db ? ;результат

**.code**

**start:**

**mov ax,@data**

**mov ds,ax**

mov al, x1 ;помещаем в al первое слагаемое

add al, x2 ;складываем х1 и х2

mov y, al ;помещаем результат на место

**mov ax,4c00h**

**int 21h**

**end start**

## Умножение двоичных чисел

***mul множитель\_1*** - операция умножения двух целых чисел без учета знака

*Алгоритм работы:*

Команда выполняет умножение двух операндов без учета знаков. Алгоритм зависит от формата операнда команды и требует явного указания местоположения только одного сомножителя, который может быть расположен в памяти или в регистре. Местоположение второго сомножителя фиксировано и зависит от размера первого сомножителя. Местоположение результата также зависит от размера первого сомножителя.

mul dl; ax:=al\*dl, dl- множитель\_1 , al- множитель\_2

mul x1; dx:ax=ax\*0ad91h, x1 word- множитель\_1 , ax- множитель\_2

mul ecx; edx:eax=eax\*ecx, ecx- множитель\_1 , eax- множитель\_2

в результате умножения может возникнуть ситуация когда результат по размеру превысит 16 или 32 бита, тогда старшая часть результата умножения заноситься в dx или edx соответственно.

***imul*** ***множитель\_1*** - операция умножения двух целочисленных двоичных значений со знаком

## Деление двоичных чисел

***div* *делитель*** - выполнение операции деления двух двоичных беззнаковых значений

*Алгоритм работы:*

Для команды необходимо задание двух операндов — делимого и делителя. Делимое задается неявно, и размер его зависит от размера делителя, который указывается в команде. Расположение результата зависит от размера делителя.

div dl ;ah:al=ax/dl, ax –делимое, dl- делитель , ah-частное, al -остаток

div x1 ;ax:dx=dx:ax/0ad91h, dx:ax –делимое, x1 word- делитель ,

;ax-частное, dx -остаток

div ecx ;eax:edx=edx:eax/ecx, edx:eax –делимое, ecx- делитель ,

;eax-частное, edx -остаток

***idiv* *делитель*** - операция деления двух двоичных значений со знаком

Пример 7. Умножение двух однобайтных чисел.

**model small**

**.stack 100h**

**.data**

x1 db 78 ;первый множитель

yl db ? ;первый байт результата

yh db ? ;второй байт результата

**.code**

**start:**

**mov ax,@data**

**mov ds,ax**

xor ax, ax ;очищаем регистр ax

mov al, 25 ;помещаем в al второй сомножитель

mul x1

jnc m1 ;если нет переполнения, переходим на метку m1

mov yh,ah ;иначе старший байт результата помещаем в yh

m1:

mov yl, al ;результат помещаем на место

**mov ax,4c00h**

**int 21h**

**end start**

Пример 8. Деление двух однобайтных чисел.

**model small**

**.stack 100h**

**.data**

x1 db 6 ;делитель

yl db ? ;остаток

yh db ? ;частное

**.code**

**start:**

**mov ax,@data**

**mov ds,ax**

xor ax, ax ;очищаем регистр ax

mov ax, 25 ;помещаем в al делимое

div x1

mov yh,ah ;помещаем частное на место

mov yl, al ;помещаем остаток на место

**mov ax,4c00h**

**int 21h**

**end start**

**ASCII коды и их интепритация**

Ввод информации с клавиатуры и вывод ее на экран осуществляется в символьном виде, т.е. любой символ предоставляется в ASCII кодах. Причем на один символ идет один ASCII код. На два символа – два ASCII кода, и т.д. Любое число, вводимое с клавиатуры и выводимое на экран, представляется последовательностью ASCII кодов.

**Табл.1. ASCII коды цифр**

|  |  |  |  |
| --- | --- | --- | --- |
| Символ шестнадцатеричной цифры | Двоичная тетрада | ASCII код  (двоичное представление) | Разница |
| 0 | 0000 | 30h (0011 0000) | 30h |
| 1 | 0001 | 31h (0011 0001) | 30h |
| 2 | 0010 | 32h (0011 0010) | 30h |
| 3 | 0011 | 33h (0011 0011) | 30h |
| 4 | 0100 | 34h (0011 0100) | 30h |
| 5 | 0101 | 35h (0011 0101) | 30h |
| 6 | 0110 | 36h (0011 0110) | 30h |
| 7 | 0111 | 37h (0011 0111) | 30h |
| 8 | 1000 | 38h (0011 1000) | 30h |
| 9 | 1001 | 39h (0011 1001) | 30h |
| A a | 1010 | 41h (0100 0001) 61h (0110 0001) | 37h 57h |
| B b | 1011 | 42h (0100 0010) 62h (0110 0010) | 37h 57h |
| C c | 1100 | 43h (0100 0011) 63h (0110 0011) | 37h 57h |
| D d | 1101 | 44h (0100 0100) 64h (0110 0100) | 37h 57h |
| E e | 1110 | 45h (0100 0101) 65h (0110 0101) | 37h 57h |
| F f | 1111 | 46h (0100 0110) 66h (0110 0110) | 37h 57h |

Рассмотрим последовательность действий для преобразования чисел в их ASCII код и наоборот.

Ввод информации с клавиатуры:

1. Ввод символа с клавиатуры, один ASCII код находится в dl. Заранее не известно, что это за число от 0 до 9 или от а до f.

Cmp dl, 040h

Jb m1 ; если ASCII код меньше 40h значит ввели цифру от 0 до 9,

;переходим на метку m1

Cmp dl, 047h ; иначе ввели букву, заглавную или маленькую?

Jb m2 ;если ввели заглавную, переходим на m2, иначе выполняем

;дальше по программе

sub dl, 057h ;в dl получаем из символов число a..f h

jmp m3 ;переходим на m3 чтобы не выполнять лишних вычислений

m2: sub dl, 037h ;в dl получаем из символов число a..f h

jmp m3

m1: sub dl, 030h ;в dl получаем из символов число 0..9 h

m3:

Далее приведен оптимизированный код преобразования числа из ASCII кодов. Подумайте в чем разница.

Cmp dl,040h

Jb m1

Cmp dl,047h

Jb m2

Sub dl, 020h

m2: sub dl, 07h

m1: sub dl, 030h

1. Ввод строки, отличается только тем, что такое сравнение надо проводить с каждым элементом, т.е. надо организовать цикл и обращение к каждому элементу. Рассмотрим позднее.

Вывод информации на экран

1. Предположим что, число, которое мы хотим вывести, находится в регистре bl. Вывод символа осуществляется из регистра dl, 02 функция INT 21H. Число может быть одно или двузначное, например 7h или 5Fh. Для универсальности программы будем считать, что надо вывести двузначное число. А для этого надо получить отдельно десятки и единицы, и получить для них два ASCII кода.

0

;двузначное число которое хотим вывести находится в bl

mov dl, bl ; помещаем число в регистр dl

;сдвигаем содержимое dl на 4 бита вправо, чтобы получить отдельно десятки

shr dl, 4

and bl, 0fh ;получаем отдельно единицы

cmp dl, 0ah ;сравниваем dl с ah

jb m1 ;если меньше переходим на m1

add dl, 07h

m1: add dl, 30h

mov ah, 02h

int 21h

mov dl, bl

cmp dl, 0ah ;сравниваем dl с ah

jb m2

add dl, 07h

m2: add dl, 30h

int 21h

Попробуйте сами разобраться в приведенном кусочке кода.

5

F

7

5

F

3

5

6

6

ASCII код 5Fh

**Команды передачи управления**

По принципу действия, команды микропроцессора, обеспечивающие организацию переходов в программе, можно разделить на три группы:

1. Команды безусловной передачи управления:

- [команда безусловного перехода](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideLessonLesson10%22%20l%20); jmp

- [вызова процедуры и возврата из процедуры](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideLessonLesson10%22%20l%20); call, ret

- вызова программных прерываний и возврата из программных прерываний. Int, iret

2. Команды [условной передачи управления](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideLessonLesson10%22%20l%20):

- [команды перехода по результату команды сравнения cmp](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideLessonLesson10%22%20l%20);

- команды [перехода по состоянию определенного флага](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideLessonLesson10%22%20l%20);

- команды [перехода по содержимому регистра ecx/cx](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideLessonLesson10%22%20l%20).

3. Команды [управления циклом](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideLessonLesson10%22%20l%20):

- команда [организации цикла со счетчиком ecx/cx](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideLessonLesson10%22%20l%20);

- команда организации цикла со счетчиком ecx/cx с возможностью [досрочного выхода из цикла по дополнительному условию](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideLessonLesson10%22%20l%20).

*jmp* адрес\_перехода - безусловный переход без сохранения информации о точке возврата. Аналог goto.

### **Условные переходы**

Команды условного перехода имеют одинаковый синтаксис:

**jcc метка\_перехода**

Мнемокод всех команд начинается с “**j**” — от слова *jump* (прыжок), *cc* — определяет конкретное условие, анализируемое командой. Что касается операнда *метка\_перехода*, то эта метка может находится только в пределах текущего сегмента кода, межсегментная передача управления в условных переходах не допускается.

Для того чтобы принять решение о том, куда будет передано управление командой условного перехода, предварительно должно быть сформировано условие, на основании которого и будет приниматься решение о передаче управления. Источниками такого условия могут быть:

* + - любая команда, изменяющая состояние арифметических флагов;
    - команда сравнения **cmp**, сравнивающая значения двух операндов;
    - состояние регистра ecx/cx.

**jcxz метка\_перехода** (Jump if cx is Zero) — переход, если cx ноль;

**jecxz** метка\_перехода (Jump Equal ecx Zero) — переход, если **ecx** ноль.

#### Условные переходы по содержимому флагов

|  |  |  |  |
| --- | --- | --- | --- |
| Название флага | Номер бита в eflags/flag | Команда условного перехода | Значение флага для осуществления перехода |
| Флаг переноса cf | 1 | jc | cf = 1 |
| Флаг четности pf | 2 | jp | pf = 1 |
| Флаг нуля zf | 6 | jz | zf = 1 |
| Флаг знака sf | 7 | js | sf = 1 |
| Флаг переполнения of | 11 | jo | of = 1 |
| Флаг переноса cf | 1 | jnc | cf = 0 |
| Флаг четности pf | 2 | jnp | pf = 0 |
| Флаг нуля zf | 6 | jnz | zf = 0 |
| Флаг знака sf | 7 | jns | sf = 0 |
| Флаг переполнения of | 11 | jno | of = 0 |

Пример 9. Определите, равны ли два числа вводимые пользователем с клавиатуры. Определить равенство чисел можно используя вычитание, если разность исследуемых чисел равна 0, то они равны.

**model small**

**.stack 100h**

**.data**

s1 db 'числа равны$'

s2 db 'числа не равны$'

.code

start:

mov ax,@data

mov ds,ax

mov ah,01h

int 21h ;ввели первое число

mov dl,al ;посылаем в dl первое число

int 21h ;ввели второе число

sub al,dl ;сравнили числа

jnz m1 ;если получили не 0 результат, то на метку m1

mov dx, offset s1 ;иначе выводим строку s1, о том что числа равны.

jmp m2

m1: mov dx, offset s2 ;числа не равны, выводим строку s2

m2: mov ah,09h

int 21h ;вывод информационную строку

**mov ax,4c00h**

**int 21h**

**end start**

**Команда сравнения cmp**

*cmp* операнд\_1,операнд\_2 - сравнивает два операнда и по результатам сравнения устанавливает флаги. Команда сравнения cmp имеет интересный принцип работы. Он абсолютно такой же, как и у команды вычитания sub. Единственное, чего она не делает — это запись результата вычитания на место первого операнда.

*Алгоритм работы:*

-выполнить вычитание (операнд1-операнд2);

-в зависимости от результата установить флаги, операнд1 и операнд2 не изменять (то есть результат не запоминать).

Условные переходы после команд сравнения

|  |  |  |  |
| --- | --- | --- | --- |
| Типы операндов | Мнемокод команды условного перехода | Критерий условного перехода | Значения флагов для осществления перехода |
| Любые | je | операнд\_1 = операнд\_2 | zf = 1 |
| Любые | jne | операнд\_1<>операнд\_2 | zf = 0 |
| Со знаком | jl/jnge | операнд\_1 < операнд\_2 | sf <> of |
| Со знаком | jle/jng | операнд\_1 <= операнд\_2 | sf <> of or zf = 1 |
| Со знаком | jg/jnle | операнд\_1 > операнд\_2 | sf = of and zf = 0 |
| Со знаком | jge/jnl | операнд\_1 => операнд\_2 | sf = of |
| Без знака | jb/jnae | операнд\_1 < операнд\_2 | cf = 1 |
| Без знака | jbe/jna | операнд\_1 <= операнд\_2 | cf = 1 or zf=1 |
| Без знака | ja/jnbe | операнд\_1 > операнд\_2 | cf = 0 and zf = 0 |
| Без знака | jae/jnb | операнд\_1 => операнд\_2 | cf = 0 |

Пример 10. Определите, равны ли два числа вводимые пользователем с клавиатуры.

**model small**

**.stack 100h**

**.data**

s1 db 'числа равны$'

s2 db 'числа не равны$'

.code

start:

mov ax,@data

mov ds,ax

mov ah,01h

int 21h ;ввели первое число

mov dl,al

mov ah,01h

int 21h ;ввели второе число

cmp al,dl ;сравнили числа

jne m1

mov dx, offset s1

jmp m2

m1: mov dx, offset s2

m2: mov ah,09h

int 21h ;вывод информационную строку

**mov ax,4c00h**

**int 21h**

##### **end start**

Пример 11. Даны три числа, найти среди них максимальное.

**model small**

**.stack 100h**

**.data**

s1 db 'максимальное число',10,13,'$'

x1 db 34

x2 db 56

x3 db 45

.code

start:

mov ax,@data

mov ds,ax

mov dx, offset s1

mov ah,09h

int 21h ;вывод информационную строку

;находим максимальное число

mov dl,x1 ;dl:=x1

cmp dl,x2 ;сравниваем х1 и х2

ja m1 ;если х1>х2, то на m1

mov dl,x2 ;иначе dl:=x2

m1: cmp dl,x3 ;сравниваем dl и х2

ja m2 ;если dl>х3 то на m2

mov dl,x3

;в dl находится самый максимальный элемент

m2: mov ah,02h

int 21h ;выводим максимальный элемент

**mov ax,4c00h**

**int 21h**

**end start**

##### **Организация циклов**

*loop* метка\_перехода (Loop) — повторить цикл

Работа команды заключается в выполнении следующих действий:

- декремента регистра ecx/cx;

- сравнения регистра ecx/cx с нулем:

- если (ecx/cx) > 0, то управление передается на метку перехода;

- если (ecx/cx) = 0, то управление передается на следующую после loop команду

Организация цикла:

mov cx, количество циклов

м1: тело цикла

loop m1

*loope/loopz* метка\_перехода (Loop till cx <> 0 or Zero Flag = 0) — повторить цикл, пока cx <> 0 или zf = 0.

*loopne/loopnz* метка\_перехода (Loop till cx <> 0 or Not Zero flag=0) — повторить цикл пока cx <> 0 или zf = 1

Недостаток команд организации цикла loop, loope/loopz и loopne/loopnz в том, что они реализуют только короткие переходы (от –128 до +127 байт).

Организация вложенных циклов:

mov cх,n ; в сх заносим количество итераций внешнего цикла

m1:

push cx

Внешний цикл

…

mov cx,n1; в сх заносим количество итераций внутреннего цикла

m2:

тело внутреннего цикла

loop m2

…

pop cx

loop m1

Пример 12. Напишите программу подсчета у=1+2+3+…+n, n не более 10000.

model small

.stack 100h

.data

yb dd ?

ym dw ?

s1 db 'введите n',10,13,'$'

.code

start:

mov ax,@data

mov ds,ax

mov dx, offset s1

mov ah,09h

int 21h

mov cx,3

m: shl bx,4

mov ah,01h

int 21h вводим n в регистр bx

sub ax,130h

add bx,ax

loop m

mov cx,bx

xor dx,dx

xor al,al

m1: add dx,cx считаем у

jnc m2

mov al,1

m2: loop m1

cmp al,1

je m3

mov ym,dx

m3: mov yb,edx

mov ax,4c00h

int 21h

## end start

# Команды обработки строк

**Цепочка** – это последовательность элементов, размер которых может быть байт, слово, двойное слово. Содержимое этих элементов может быть любое – символы, числа. В системе команд микропроцессора имеется семь *операций-примитивов* обработки цепочек. Каждая из них реализуется в микропроцессоре тремя командами, в свою очередь, каждая из этих команд работает с соответствующим размером элемента — байтом, словом или двойным словом.

*Типовой набор действий* для выполнения любой цепочечной команды:

* Установить значение флага df в зависимости от того, в каком направлении будут обрабатываться элементы цепочки — в направлении возрастания или убывания адресов.
* Загрузить указатели на адреса цепочек в памяти в пары регистров ds:(e)si и es: (e)di.
* Загрузить в регистр ecx/cx количество элементов, подлежащих обработке.
* Выдать цепочечную команду с префиксом повторений.

Пересылка цепочек

[**movs**](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideLessonLesson11%22%20l%20) **адрес\_прием, адрес\_источника** (MOVe String)- переслать цепочку;

[**movsb**](D:DASHKALibrari%20ot%20DariaAssemblerAssemblerguideTextCommand.htm#movs) MOVe String Byte) — переслать цепочку байт;

***movsw*** (MOVe String Word) — переслать цепочку слов;

***movsd*** (MOVe String Double word) — переслать цепочку двойных слов.

Команда копирует байт, слово или двойное слово из цепочки источника, в цепочку приемника. Размер пересылаемых элементов ассемблер определяет, исходя из атрибутов идентификаторов. К примеру, если эти идентификаторы были определены директивой db, то пересылаться будут байты, если идентификаторы были определены с помощью директивы dd, то пересылке подлежат двойные слова.

Для цепочечных команд с операндами типа **movs адрес\_приемника,адрес\_источника**, не существует машинного аналога. При трансляции в зависимости от типа операндов транслятор преобразует ее в одну из трех машинных команд: **movsb, movsw** или **movsd**.

Сама по себе команда movs пересылает только один элемент, исходя из его типа, и модифицирует значения регистров esi/si и edi/di. Если перед командой написать префикс **rep**, то одной командой можно переслать до 64 Кбайт данных. Число пересылаемых элементов должно быть загружено в *счетчик* — регистр **cx** (use16) или **ecx** (use32).

Пример 13. Пересылка строк командой movs

MODEL small

.STACK 256

.data

source db 'Тестируемая строка','$' ;строка-источник

dest db 19 DUP (' ') ;строка-приёмник

.code

main:

mov ax,@data ;загрузка сегментных регистров

mov ds,ax ;настройка регистров DS и ES на адрес сегмента данных

mov es,ax

cld ;сброс флага DF — обработка строки от начала к концу

lea si,source ;загрузка в si смещения строки-источника

lea di,dest ;загрузка в DS смещения строки-приёмника

mov cx,20 ;для префикса rep — счетчик повторений (длина строки)

rep movs dest,source ;пересылка строки

lea dx,dest

mov ah,09h ;вывод на экран строки-приёмника

int 21h

mov ax,4c00h

int 21h

end main

***Операция сравнения цепочек***

***cmps адрес\_приемника,адрес\_источника***(CoMPare String) — сравнить строки;

***cmpsb*** (CoMPare String Byte) — сравнить строку байт;

***cmpsw*** (CoMPare String Word) — сравнить строку слов;

***cmpsd*** (CoMPare String Double word) — сравнить строку двойных слов.

Алгоритм работы команды cmps заключается в последовательном выполнении вычитания (элемент цепочки-источника — элемент цепочки-получателя) над очередными элементами обеих цепочек. Принцип выполнения вычитания командой cmps аналогичен команде сравнения cmp. Она, так же, как и cmp, производит вычитание элементов, не записывая при этом результата, и устанавливает флаги zf, sf и of.

После выполнения вычитания очередных элементов цепочек командой cmps, индексные регистры esi/si и edi/di автоматически изменяются в соответствии со значением флага df на значение, равное размеру элемента сравниваемых цепочек.

***Операция сканирования цепочек***

***scas адрес\_приемника*** (SCAning String) — сканировать цепочку;

***scasb*** (SCAning String Byte) — сканировать цепочку байт;

***scasw*** (SCAning String Word) — сканировать цепочку слов;

***scasd*** (SCAning String Double Word) — сканировать цепочку двойных слов

Эти команды осуществляют поиск искомого значения, которое находится в регистре al/ax/eax. Принцип поиска тот же, что и в команде сравнения cmps, то есть последовательное выполнение вычитания

(содержимое регистра\_аккумулятора – содержимое очередного\_элемента\_цепочки).

В зависимости от результатов вычитания производится установка флагов, при этом сами операнды не изменяются.

***Загрузка элемента цепочки в аккумулятор***

***lods адрес\_источника*** (LOaD String) — загрузить элемент из цепочки в регистр-аккумулятор al/ax/eax;   
***lodsb***(LOaD String Byte) — загрузить байт из цепочки в регистр al;

***lodsw*** (LOaD String Word) — загрузить слово из цепочки в регистр ax;

***lodsd*** (LOaD String Double Word) — загрузить двойное слово из цепочки в регистр eax.

Эта операция-примитив позволяет извлечь элемент цепочки и поместить его в регистр-аккумулятор al, ax или eax. Эту операцию удобно использовать вместе с поиском (сканированием) с тем, чтобы, найдя нужный элемент, извлечь его (например, для изменения).

***Перенос элемента из аккумулятора в цепочку***

**stos адрес\_приемника**(STOre String) - сохранить элемент из регистра-аккумулятора al/ax/eax в цепочке;

**stosb *(STOre String Byte)*** - сохранить байт из регистра al в цепочке;

**stosw** ***(STOre String Word)*** - сохранить слово из регистра ax в цепочке;

**stosd *(STOre String Double Word)*** - сохранить двойное слово из регистра eax в цепочке.

Эта операция-примитив позволяет произвести действие, обратное команде lods, то есть сохранить значение из регистра-аккумулятора в элементе цепочки. Эту операцию удобно использовать вместе с операцией поиска (сканирования) scans и загрузки lods, с тем, чтобы, *найдя нужный элемент, извлечь его в регистр и записать на его место новое значение*.

**Пример 14. Подсчитайте количество несовпадающих элементов в заданной и введенной строках.**

Model small

.stack 100h

.data

s0 db ‘Заданная строка$’

s1 db 16 ;задаем количество символов во вводимой строке + знак Enters

s2 db ?, 16 dup (?) ; ?- под количество введенных символов, массив под строку

s3 db 10,13, ‘Количество несовпадающих элементов - $' ;информац. строка

.code

mov ax, @data

mov ds, ax ;задаем адрес сегмента данных

mov es, ax ;настраиваем адрес сегмента данных, где хранится строка приемник

;вводим сравниваемую строку

mov ah, 0ah

mov dx, offset s1

int 21h

;выводим информационную строку

mov ah, 09h

mov dx, offset s3

int 21h

;сравниваем строки, один элемент из заданной строки сравниваем со всеми ; элементами введенной строки

mov dl,’0’ ;в dl ascii-код 0

mov cx, 16 ;в сх количество элементов в заданной строке

mov si, offset s0 ;в si адрес заданной строки-источника

z\_str: push cx ;сохраняем счетчик внешних циклов в стеке

lodsb ;загружаем элемент из заданной строки в аккумулятор, al

mov di, offset s2 ;в di адрес введенной строки-приемника

mov cl, s2[di] ;в cl количество введенных элементов

xor ch, ch ; обнуляем ch, т.к. в цикле счетчиком является сх

inc di ;на первый элемент строки-приемника

repe scacb

;сканируем строку-приемник до тех пор пока элемент не = содержимому al,

;или пока не кончится строка

jz m1 ;zf=1, если в строке встретился элемент = содержимому al

inc dl ; считаем количество не совпадающих элементов

m1: ;внутренний цикл по введенной строке закончился

pop cx ;восстанавливаем содержимое сх

loop z\_str

;после выхода из цикла в dl количество не совпадающих элементов

mov ah, 02h

int 21h ;выводим dl

mov ax,4c00h

int 21h

end

**Массивы**

Организация одномерных массивов

**Все элементы массива располагаются в памяти последовательно**

**Описание элементов массива**

mas db 1,2,3,4,5

mas dw 5 dup (0)

#### Доступ к элементам массива

mov ax,mas[si] ; в si номер элемента в массиве

mov mas[si], ax ; в di номер элемента в массиве

Пример 15. Найти в строке хотя бы один нулевой элемент

model small

.stack 100h

.data

bufer dw 25 ;формирую размер буфера для ввода строки

mas db 25 dup (' ') ;формирую буфер

subj1 db ‘в строке найден нулевой элемент', '$'

subj2 db ‘в строке не найден нулевой элемент', '$'

.code

main:

mov ax,@data

mov ds,ax

; ввод строки с клавиатуры

mov ah,0ah

mov dx, offset bufer

int 21h

;поиск нулевого элемента

xor si, si

mov cl, mas[si] ;загружаем в сх количество элементов в строке

mov al, 030h ;в ax загружаем ASCII код нуля

m1: inc si

cmp al, mas[si]

je m2

;если в строке найдем нулевой элемент, то выходим из цикла на вывод subj1

loop m1

;нормальный выход из цикла означает что в строке нет нулевых элементов

lea dx, subj2

jmp m3

m2: lea dx,subj1

m3: mov ah, 09h

int 21h

mov ax,4c00h

int 21h

end main

**Организация двумерных массивов**

**!Специальных средств для описания двумерных массивов в ассемблере нет!**

Двумерный массив описывается также как и одномерный массив, отличие заключается в трактовке расположения элементов. Пусть последовательность элементов трактуется как двумерный массив, расположенный по строкам, тогда адрес элемента [i,j] вычисляется так

База+колич\_элем\_строке\*размер\_элем\*I+j

Пример 16. Найти максимальный элементы в каждой строке массива 5\*7

model small

.stack 100h

.data

mas dw 5 dup( 7 dup(0))

max dw 0

subj db ‘введите строку',13,10,'$'

.code

main:

mov ax, @data

mov ds, ax

;заполнение массива

xor si, si

mov cx, 05h

incykl: push cx

mov ah, 09h

lea dx, subj

int 21h ;вывод информационной строки

mov cx, 07h

mov ah, 01h

outcykl: int 21h ;ввод элементов массива

mov mas[si], ax ;размещение элементов на месте

inc si

inc si

loop outcykl

pop cx

loop incykl

;поиск максимального/ минимального в строках

xor si,si

mov cx, 05h

s1t: push cx

mov cx, 06h

mov dx, mas[si]

maxi: add si, 2

cmp dx, mas[si]

ja min1 ;если меньше то переходим

mov dx, mas[si]

min1: loop maxi

;вывод максимального

mov ah, 02h

int 21h

pop cx

loop s1t

mov ax, 04c00h

int 21h

end main

**Процедуры. Макрокоманды**

***Процедура***, часто называемая подпрограммой, - это правильным образом оформленная совокупность команд, которая будучи однократно описана, при необходимости может быть вызвана в любом месте программы. Процедура представляет собой группу команд для решения конкретной подзадачи и обладает средствами получения управления из точки вызова задачи более высокого уровня и возврата управления в эту точку. В простейшем случае программа может состоять из одной процедуры.

Описание процедуры может размещается в любом месте программы, но таким образом чтобы на нее случайным образом не попало управление:

* + - в начале программы, до первой исполняемой команды;
    - в конце, после команды возвращающей управление операционной системе;
    - промежуточный вариант, тело процедуры располагается внутри другой процедуры или основной программы. В этом случае необходимо предусмотреть обход процедуры командой jmp;
    - в другом модуле.

Синтаксис описания процедуры:

*Имя\_процедуры PROC* заголовок

Команды, директивы тело процедуры

*[ret]* возврат из процедуры

*[имя\_процедуры] ENDP* конец процедуры

Вызов процедуры осуществляется командой

*CALL [модификатор] имя\_процедуры*

Команда call передает управление по адресу с символическим адресом имя\_процедуры, с сохранением в стеке адреса возврата, команды следующей после команды call.

Возврат из процедуры осуществляется по команде

*RET [число]*

Команда ret считывает адрес возврата из стека и загружает его в регистры cs и ip/eip, возвращая таким образом управление команде, следующей за командой call. Число – необязательный параметр, обозначающий количество элементов, удаляемых из стека при возврате из процедуры. Размер элемента зависит от используемой модели сегментации 32 или 16 разрядной.

Передача аргументов из/в процедуру может осуществляться через регистры, переменные или стек.

**Пример.**

|  |  |
| --- | --- |
| Model small  .stack 100h  .data  w db 25 dup (?)  .code  vvod proc  mov ah, 0ah  lea dx, w  int 21h  ret  vvod endp  main: | …  Call schet  Call vvod  …  exit:  mov ax,4c00h  int 21h  schet proc  ..  ret  schet endp  end main |

Макрокоманда является одним из многих механизмов замены текста программы. С помощью макрокоманды в текст программы можно вставлять последовательности строк и привязывать их к месту вставки. **Макрокоманда** представляет собой строку, содержащую некоторое имя – имя макрокоманды, предназначенное для того, чтобы быть замещенным одной или несколькими другими строками при трансляции.

Для работы с макрокомандой вначале необходимо задать ее шаблон-описание, так называемое макроопределение.

Имя\_макрокоманды MACRO [список\_формальных\_аргументов]

<Тело макроопределения>

ENDM

Существует три варианта расположения макроопределений:

* + - в начале исходного текста программы до сегмента кода и данных с тем, чтобы не ухудшать читабельность программы. В данном случае макрокоманды будут актуальны только в пределах этой программы;
    - в отдельном файле. Для того, чтобы использовать эти макроопределения в других программах, необходимо в начале исходного текста этих программ записать директиву

include имя\_файла

* + - в макробиблиотеке. Макробиблиотека создается в том случае, когда написанные макросы используются практически во всех программах. Подключается библиотека директивой include. Недостаток этого и предыдущего методов в том, что в исходный текст программы включаются абсолютно все макроопределения. Для исправления ситуации можно использовать директиву purge, в качестве операндов которой перечисляются макрокоманды, которые не должны включаться в текст программы.

Include macrobibl.inc ;в исходный текст программы будут вставлены строки из macrobibl.inc

Purge outstr, exit ;за исключением макроопределений outstr, exit

Активизация макроса осуществляется следующим образом:

*Имя\_макрокоманды список\_ фактических\_ аргументов*

|  |  |
| --- | --- |
| Model small  Vivod macro rg  Mov dl, rg  Mov ah, 02h  Int 21h  endm  .data  ..  .code  ..  vivod al  .. | Model small  sravnenie macro rg, met  cmp rg , ‘a’  ja met  add rg, 07h  met: add rg, 30h  endm  .data  ..  .code  ..  sravnenie al, m1.. |

Функционально макроопределения похожи на процедуры. Сходство их в том, что и те, и другие достаточно один раз где-то описать, а затем вызывать их специальным образом. На этом их сходство заканчивается, и начинаются различия, которые в зависимости от целевой установки можно рассматривать и как достоинства и как недостатки:

- в отличие от процедуры, текст которой неизменен, макроопределение в процессе макрогенерации может меняться в соответствии с набором фактических параметров. При этом коррекции могут подвергаться как операнды команд, так и сами команды. Процедуры в этом отношении объекты менее гибки;

- при каждом вызове макрокоманды ее текст в виде макрорасширения вставляется в программу. При вызове процедуры микропроцессор осуществляет передачу управления на начало процедуры, находящейся в некоторой области памяти в одном экземпляре. Код в этом случае получается более компактным, хотя быстродействие несколько снижается за счет необходимости осуществления переходов.

Пример 17. Найти максимальный элементы в каждой строке массива 5\*7, с использованием процедур

model small

.stack 100h

.data

mas dw 5 dup( 7 dup(0))

max dw 0

subj db ‘введите строку',13,10,'$'

.code

;процедура ввода строки

**vvod\_str proc**

mov ah, 09h

lea dx, subj

int 21h

mov cx, 07h

mov ah, 01h

outcykl: int 21h

mov mas[si], ax

inc si

inc si

loop outcykl

**ret**

**vvod\_str endp**

;процедура поиска максимального в строке

**poick­\_­maxi proc**

mov cx, 06h

mov dx, mas[si]

maxi: add si, 2

cmp dx, mas[si]

ja min1 ;если меньше то переходим

mov dx, mas[si]

min1: loop maxi

**ret**

**poick­\_­maxi endp**

**main** **proc**

mov ax, @data

mov ds, ax

xor si, si ;заполнение массива

mov cx, 05h

incykl: push cx

**call vvod\_str**  ;вызов процедуры по вводу строки

pop cx

loop incykl

;поиск максимального/ минимального в строках

xor si,si

mov cx, 05h

s1t: push cx

**call poick­\_­maxi** ;вызов процедуры поиска максимального элемента

mov ah, 02h ;вывод максимального

int 21h

pop cx

loop s1t

mov ax, 04c00h

int 21h

**endр main**

**Программирование контроллера приоритетных прерываний**

Для организации обработки аппаратных прерываний в вычислительных системах применяется программируемый контроллер прерываний, выполненный в виде специальной микросхемы i8259А, отечественный аналог микросхема КР580ВМ59. Эта микросхема может обрабатывать запросы от восьми источников внешних прерываний. В стандартной конфигурации вычислительных систем используют две последовательно соединенные микросхемы i8259А.

Функции микросхемы ПКП:

* фиксирование запросов на обработку прерывания от восьми источников, формирование единого запроса на прерывание и подача его на вход INTR микропроцессора;
* формирование номера вектора прерывания и выдача его на шину данных;
* организация приоритетной обработки прерываний;
* запрещение (маскирование) прерываний с определенными номерами.

На рис.1 представлена структурная схема контроллера.

**Рис.1. Структурная схема ПКПП.**

Irq1

Irq2

Irq3

Irq4

Irq5

Irq7

Irq6

Irq0

INTA

INT

Блок управления чтением/записью

Буфер данных

Схема каскадирования

d7 ... d0

Регистр состояния (ISR)

Арбитр приоритетов

Регистр маскирования прерываний (IMR)

Регистр прерываний (IRR)

Программируемый контроллер прерываний (ПКП)

Схема управления ПКП

# Программирование контроллера приоритетных прерываний

***Цель работы:***

Исследование принципа программного управления микросхемы контроллера прерываний (ПКП) i8259А с помощью ПК, исследование различных режимов работы ПКП

Микросхема i8259A имеет два состояния:

* состояние *настройки* параметров обслуживания прерываний, во время которого путем посылки в определенном порядке так называемых *управляющих слов* производится инициализация контроллера;
* состояние *работы —* это обычное состояние контроллера, в котором производится фиксация запросов на прерывание и формирование управляющей информации для микропроцессора в соответствии с параметрами настройки.

Возможность программирования контроллера позволяет достаточно гибко изменять алгоритмы обработки аппаратных прерываний.

В процессе загрузки компьютера и в дальнейшем во время работы контроллер прерываний настраивается на работу в одном из шести режимов:

**1.Режим фиксированных приоритетов (Fixed Priority, Fully Nested Mode).**

В этом режиме контроллер находится сразу после инициализации. Запросы прерываний имеют жесткие приоритеты от 0 до 7 (0 - высший) и обрабатываются в соответствии с приоритетами. Прерывание с меньшим приоритетом никогда не будет обработано, если в процессе обработки прерываний с более высокими приоритетами постоянно возникают запросы на эти прерывания.

**2. Автоматический сдвиг приоритетов (Automatic Rotation). Режим циклической обработки прерываний.**

В этом режиме дается возможность обработать прерывания всех уровней без их дискриминации. Например, после обработки прерывания уровня 4 ему автоматически присваивается низший приоритет, при этом приоритеты для всех остальных уровней циклически сдвигаются и прерывания уровня 5 будут иметь в данной ситуации высший приоритет и, следовательно, возможность быть обработанными.

**3. Программно-управляемый сдвиг приоритетов (Specific Rotation).**

Программист может сам передать команду циклического сдвига приоритетов ПКП, задав соответствующее управляющее слово. В команде задается номер уровня, которому требуется присвоить максимальный приоритет. После выполнения такой команды устройство работает так же, как и в режиме фиксированных приоритетов, с учетом их сдвига. Приоритеты сдвигаются циклически, таким образом если максимальный приоритет был назначен уровню 3, то уровень 2 получит минимальный и будет обрабатываться последним.

**4 Автоматическое завершение обработки прерывания (Automatic End Of Interrupt, AEOI).**

В обычном режиме работы процедура обработки аппаратного прерывания должна перед своим завершением очистить свой бит в ISR специальной командой, иначе новые прерывания не будут обрабатываться ПКП. В режиме AEOI нужный бит в ISR автоматически сбрасывается в тот момент, когда начинается обработка прерывания нужной процедурой обработки и от нее не требуется издавать команду завершения обработки прерывания (EOI). Сложность работы в данном режиме обуславливается тем, что все процедуры обработки аппаратных прерываний должны быть повторно входимыми, т. к. за время их работы могут повторно возникнуть прерывания того же уровня.

**5. Режим специальной маски (Special Mask Mode).**

Данный режим позволяет отменить приоритетное упорядочение обработки запросов и обрабатывать их по мере поступления. После отмены режима специальной маски предшествующий порядок приоритетов уровней сохранается.

**6. Режим опроса (Polling Mode).**

В этом режиме аппаратные прерывания не происходят автоматически. Появление запросов на прерывание должно определяться считыванием IRR. Данный режим позволяет так же получить от ПКП информацию о наличии запросов на прерывания и, если запросы имеются, номер уровня с максимальным приоритетом, по которому есть запрос

***Программирование контроллера прерываний i8259A***

Для вывода информации в ПКП используются 2 порта ввода-вывода. Порт с четным адресом (обычно это порт 20h) и порт с нечетным адресом (обычно 21h). Через эти порты могут быть переданы 4 слова инициализации (Initialization Control Word, ICW1 - ICW4), задающие режим работы ПКП, и 3 операционных управляющих слова (слова рабочих приказов, Operation Control Words, OCW1 - OCW3).

В порт с четным адресом выводятся ICW1, OCW2 и OCW3.

Порт с нечетным адресом используется для вывода ICW2, ICW3, ICW4 и OCW1. Неоднозначности интерпретации данных в этом случае так же не возникает, т. к. слова инициализации ICW2 - ICW4 должны непосредственно следовать за ICW1, выведенным в порт с четным адресом и выводить в промежутке между ними OCW1 не следует, оно не будет опознано контроллером.

Выводом в порт с четным адресом управляющего слова инициализации ICW1 начинается инициализация ПКП. В процессе инициализации контроллер последовательно принимает управляющие слова ICW1 - ICW4. При наличии в системе одного контроллера ICW3 не выводится. Наличие ICW4 определяется содержанием ICW1. При наличии каскада из нескольких ПКП каждый из них инициализируется отдельно.

Для инициализации и управления работой ведомого контроллера используются адреса A0h, A1h. В порт с адресом A0h выводятся ICW1, OCW2 и OCW3. Порт с адресом A1h используется для вывода ICW2, ICW3, ICW4 и OCW1.

При наличии в системе ведомого контроллера слово ICW3 для контроллеров ПКП обязательно.

**Формат ICW1 следующий:**

|  |  |
| --- | --- |
| **Биты ICW1** | **Назначение и содержание** |
| **0** | 1 – управляющее слово ICW4 будет присутствовать в данной последовательности приказов |
| **1** | 0 – каскадное подключение ПКП (ICW3 будет в последовательности)  1 – одиночное подключение ПКП (ICW3 не будет) |
| **2** | 0 – не используется |
| **3** | 0 – прерывание по перепаду сигнала |
| **4** | 1 – признак ICW1 |
| **5..7** | 0 – не используется |

**ICW2 – определение базового адреса:**

|  |  |
| --- | --- |
| **Биты ICW2** | **Назначение и содержание** |
| **0..2** | 0 – не используется |
| **3..7** | Бит для задания номера базового вектора |

Управляющее слово ICW2 задает номер вектора прерывания, процедуры обработки прерываний, для аппаратного прерывания irq0. Вектора обработки аппаратных прерываний располагаются последовательно с адреса 08h, загружаемого в начале работы процессора. Некорректное изменение номера вектора приведет к сбою всей системы.

**Формат ICW3 для ведущего контроллера следующий:**

|  |  |
| --- | --- |
| **Биты ICW3** | **Назначение и содержание** |
| **0..7** | 1- если ко входу irqN подключен ведомый  0- если ко входу irqN подключено внешнее устройство |

**Формат ICW3 для ведомого контроллера следующий:**

|  |  |
| --- | --- |
| **Биты ICW3** | **Назначение и содержание** |
| **0..3** | Задает номер уровня, на котором работает ведомый контроллер |
| **4..7** | 0 – не используется |

**Формат ICW4:**

|  |  |
| --- | --- |
| **Биты ICW4** | **Назначение и содержание** |
| **0** | Тип микропроцессора: 0 – i8080; 1 – i80x86, Pentium |
| **1** | 1. режим автоматического завершения обработки прерывания, описанный выше   0- действует обычное соглашение: процедура обработки аппаратного прерывания должна сама сбрасывать свой бит в ISR. |
| **2** | 1 – данный контроллер ведущий,  0 - данный контроллер ведомый |
| **3** | 1 – системная шина буферизована  0 - системная шина не буферизована |
| **4** | 0 - устанавливает специальный вложенный режим, применяемый при каскадировании для определения приоритетов запросов от разных контроллеров (Special Fully Nested Mode) |
| **5..7** | 0 |

В процессе работы с ПКП вы можете без переинициализации:

* маскировать и размаскировать аппаратные прерывания;
* изменять приоритеты уровней;
* издавать команду завершения обработки аппаратного прерывания;
* устанавливать/сбрасывать режим специальной маски;
* переводить контроллер в режим опроса и считывать состояние регистров ISR и IRR.

Для этого Вам потребуется вывести в порты ПКП одно из трех слов рабочих приказов OCW1 - OCW3.

**Формат OCW1 – управление регистром масок IMR:**

|  |  |
| --- | --- |
| **Биты ОCW1** | **Назначение и содержание** |
| **0..7** | 0 – разрешить прерывания уровня N  1 - запретить прерывания уровня N |

**Формат OCW2 – управление приоритетом:**

|  |  |
| --- | --- |
| **Биты ОCW2** | **Назначение и содержание** |
| **0..2** | 000-nnn – код уровня запроса irq для действий, определяемых разрядами 5-7 |
| **3..4** | 00 – признак OCW2 |
| **5** | 1. режим автоматического EOI 2. режим неавтоматического EOI |
| **6..7** | Задают операцию в сочетании с 5-м битом:  000 – автоматический режим приоритетов с автоматическим EOI  001 – сброс бита с максимальным приоритетом в ISR  011 – сброс бита в ISR для уровня с кодом nnn  100 – установка режима циклической смены приоритета при автоматическом EOI  101 – установка режима циклической смены приоритета при неавтоматическом EOI  111 - установка режима циклической смены приоритета но относительно бита nnn |

**Формат OCW3 – общее управление контроллером:**

|  |  |
| --- | --- |
| **Биты ОCW3** | **Назначение и содержание** |
| **0..1** | 10 – прочитать содержимое IRR (следующей командой из порта 020h);  11– прочитать содержимое ISR (следующей командой из порта 020h);  Содержимое IМR доступно постоянно как содержимое порта 021h. |
| **2** | 1 - переводит контроллер в режим опроса |
| **3..4** | Признак OCW3 |
| **5..6** | 11 – установить режим специального маскирования  10 – сбросить режим специального маскирования  00 или 01 – ничего не менять |
| **7** | 0 |

Распределение и приоритеты аппаратных прерываний в архитектуре АТ

|  |  |  |  |
| --- | --- | --- | --- |
| **Уровень** | **Контроллер** | **Источник прерываний** | **Приоритет уровня** |
| Irq0 | Ведущий | Таймер | 2 |
| Irq1 | Ведущий | Клавиатура | 3 |
| Irq2 | Ведущий | Выход INT ведомого |  |
| Irq8 | Ведомый | Часы реального времени | 4 |
| Irq9 | Ведомый | Вход для устройства расширения | 5 |
| Irq10 | Ведомый | Вход для устройства расширения | 6 |
| Irq11 | Ведомый | Вход для устройства расширения | 7 |
| Irq12 | Ведомый | Вход для устройства расширения | 8 |
| Irq13 | Ведомый | Ошибка сопроцессора | 9 |
| Irq14 | Ведомый | Контроллер жесткого диска | 10 |
| Irq15 | Ведомый | Вход для устройства расширения | 11 |
| Irq3 | Ведущий | Вход для устройства расширения (последовательный порт СОМ2) | 12 |
| Irq4 | Ведущий | Вход для устройства расширения (последовательный порт СОМ1) | 13 |
| Irq5 | Ведущий | Вход для устройства расширения (параллельный порт LPT2) | 14 |
| Irq6 | Ведущий | Контроллер гибкого диска | 15 |
| Irq7 | Ведущий | Вход для устройства расширения (параллельный порт LPT1) | 16 |

**Программирование контроллера прямого доступа памяти**

### **Программирование контроллера ПДП**

***Цель работы:***

Исследование принципа программного управления микросхемы, контроллера прямого доступа памяти (ПДП) i8237А с помощью ПК, исследование различных режимов работы ПДП.

**Прямой доступ к памяти – DMA (Direct Memory Access)** метод обмена данными периферийного устройства с памятью без участия процессора. В режиме прямого доступа к памяти процессор инициализирует контроллер прямого доступа к памяти – задает начальный адрес, счетчик и режим обмена, после чего освобождается. Сам обмен производит контроллером ПДП, что обеспечивает высокоскоростной обмен данными между устройствами ввода-вывода и ОЗУ без использования центрального процессора, это позволяет освободить процессор для выполнения вычислений параллельно с обменом и независимо от него. Наиболее часто возможности ПДП используются при работе с дисковыми накопителями, однако реализовано использование ПДП рядом других устройств. Ощутимые преимущества дает использование ПДП в процессе обмена с устройствами, принимающими или передающими данные достаточно большими порциями с высокой скоростью.

Четырехканальный контроллер ПДП i8237А имеет 16-разрядные регистры адреса и счетчики, что обеспечивает возможность программирования передачи блока данных размером до 64 Кбайт. Для обеспечения доступности адресного пространства памяти размером в 1 Мбайт применили внешние 4-разрядные регистры страниц DMA, отдельные для каждого канала. В этих регистрах хранятся биты адреса А[19:16], а битами А[15:0] управляет контроллер.

Микросхема i8237А допускает каскадирование при довольно гибком конфигурировании.

##### **Принципы работы контроллера ПДП**

В работе ПДП различаются 2 главных цикла: цикл ожидания (Idle cycle) и активный цикл (Active cycle). Каждый цикл подразделяется на ряд состояний, занимающих по времени один период времени (тик). Из цикла ожидания контроллер может быть переведен в состояние программирования (Program Condition) путем подачи на вход RESET сигнала высокого уровня, длительностью не менее 300 нc и следующей за ним подачи сигнала низкого уровня (уровня 0) на вывод CS (Chip Select). В состоянии программирования контроллер будет находится до тех пор, пока на выводе CS сохранится сигнал низкого уровня. В процессе программирования контроллеру задаются:

* начальный адрес памяти для обмена;
* уменьшенное на единицу число передаваемых байтов;
* направление обмена;
* требуемые режимы работы (разрешить или запретить циклическое изменение приоритетов, автоинициализацию, задать направление изменения адреса при обмене и т. д.).

Загрузка 16-разрядных регистров контроллера осуществляется через 8-разрядные порты ввода-вывода. Перед загрузкой первого (младшего) байта должен быть сброшен (очищен) триггер-защелка (триггер первый/последний, First/Last flip-flop), который изменяет свое состояние после вывода в порт первого байта и таким образом дает возможность следующей командой вывода в тот же порт загрузить старший байт соответствующего регистра.

Запрограммированный канал должен быть демаскирован (бит маски канала устанавливается при этом в 0), после чего он может принимать сигналы «Запрос на ПДП», генерируемые тем внешним устройством, которое обслуживается через этот канал. Сигнал «Запрос на ПДП» может быть также инициирован установкой в 1 бита запроса данного канала в регистре запросов контроллера. После появления сигнала запроса контроллер входит в активный цикл, в котором выполняется обмен данными. Обмен может осуществляется в одном из четырех режимов:

**1. Режим одиночной передачи (Signle Transfer Mode).**

После каждого цикла передачи контроллер освобождает шину процессору, но сразу же начинает проверку сигналов запроса и, как только обнаруживает активный сигнал запроса, инициирует следующий цикл передачи.

**2. Режим блочной передачи (Block Transfer Mode).**

В этом режиме наличие сигнала запроса требуется только до момента выдачи контроллером сигнала «Подтверждение запроса на ПДП» (DACK), после чего шина не освобождается вплоть до завершения передачи всего блока.

**3. Режим передачи по требованию (Demand Transfer Mode).**

Данный режим является промежуточным между двумя первыми: передача идет непрерывно до тех пор, пока активен сигнал запроса, состояние которого проверяется после каждого цикла передачи. Как только устройство не может продолжить передачу, сигнал запроса сбрасывается им и контроллер приостанавливает работу. Этот режим применяется для обмена с медленными устройствами, не позволяющими по своим временным характеристикам работать с ПДП в режиме блочной передачи.

**4. Каскадный режим (Cascade Mode).**

Режим позволяет включить в подсистему ПДП более одного контроллера в тех случаях, когда недостаточно четырех каналов ПДП. В этом режиме один из каналов ведущего контроллера используется для каскадирования с контроллером второго уровня. Для работы в каскаде сигнал HRQ («Запрос на захват») ведомого контроллера подается на вход DREG («Запрос на канал ПДП») ведущего, а сигнал DACK («Подтверждение запроса») ведущего подается на вход HDLA («Подтверждение захвата») ведомого.

Такая схема подключения аналогична подключению ведущего (первого) контроллера к микропроцессору, с которым он обменивается сигналами HRQ и HDLA.

**Типы возможных режимов передач**

**1. Передача память-память (Memory-to-memory DMA)**

Используется для передачи блока данных из одного места памяти в другое. Исходный адрес определяется в регистрах нулевого канала, выходной - в регистрах первого канала. Число циклов обмена (число байт минус 1) задается в регистре числа циклов канала 1. Передача происходит с использованием рабочего регистра контроллера в качестве промежуточного звена для хранения информации. При передачe память-память может быть задан специальный режим фиксации адреса (Address hold), при котором значение текущего адреса в регистре нулевого канала не изменяется, при этом весь выходной блок памяти заполняется одним и тем же элементом данных, находящимся по заданному адресу.

**2. Автоинициализация (автозагрузка, Autoinitialization)**

После завершения обычной передачи использованный канал ПДП маскируется и должен быть перепрограммирован для дальнейшей работы с ним. При автоинициализации маскировка канала после окончания передачи не происходит, а регистры текущего адреса и счетчик циклов автоматически загружаются из соответствующих регистров с начальными значениями. Таким образом для продолжения (повторения) обмена достаточно выставить сигнал запроса на ПДП по данному каналу.

**3. Режим фиксированных приоритетов**

В этом режиме канал 0 всегда имеет максимальный приоритет, а канал 3 - минимальный. Это означает, что любая передача по каналу с более высоким приоритетом будет выполняться раньше, чем по каналу с более низким приоритетом.

**4. Циклический сдвиг приоритетов**

Позволяет избежать «забивания» шины одним каналом при одновременной передаче по нескольким каналам. Каждому каналу, по которому прошла передача, автоматически присваивается низший приоритет, после чего право на передачу получает канал с наивысшим приоритетом, для которого передача в данный момент возможна. Таким образом, если в начале работы распределение приоритетов было обычным (канал 0 - наивысший), и пришли сигналы запроса на ПДП по 1-му и 2-му каналам, то сначала будет выполняться передача по первому каналу, затем он получит низший приоритет (а канал 2, соответственно, высший, т. к. сдвиг приоритетов циклический) и передача выполнится по 2-му каналу, который затем получит низший приоритет, а высший приоритет получит, соответственно, канал 3, который и будет обладать преимущественным правом на передачу.

**5. Сжатие времени передачи (Compressed transfer timing).**

В случае, если временные характеристики быстродействия обменивающихся устройств совпадают, ПДП может сократить время выполнения каждого такта передачи на 2 цикла часов за счет тактов ожидания, входящих в каждый цикл передачи.

## Распределение каналов прямого доступа

Прямой доступ к памяти был использован еще в PC/XT, где для этого применялась микросхема четырехканального контроллера 8237А.

Из четырех каналов DMA XT на шине ISA доступны только три (1, 2 и 3). Канал 0 используется для регенерации динамической памяти, и от него на шину ISA выводится только сигнал подтверждения DACKO#, он же REFRESH#. Этот сигнал может использоваться для регенерации динамической памяти, если таковая используется на платах адаптера. Адрес регенерируемой строки берется с линий адреса шины ISA. Каналы 1, 2 и 3 обеспечивают побайтную передачу данных и называются 8-битными каналами DMA.

В архитектуре AT подсистему DMA расширили, добавив второй контроллер 8237А. Его подключили к шине адреса со смещением на 1 бит, и его 16-битные регистры адреса способны управлять линиями адреса А[1б:1], младший бит адреса АО всегда нулевой. Таким образом, второй контроллер может обеспечивать передачу данных только пословно (по два байта), за что его каналы и названы 16-битными. За один сеанс второй контроллер способен передать массив до 64К 16-разрядных слов. Регистры страниц для всех каналов DMA у AT расширены до 8 бит, что делает доступной для любого канала область памяти размером 16 Мбайт (0-FFFFFFh). Стандартное назначение каналов приведено в табл. 1.

Кроме увеличения числа каналов в AT ввели дополнительную возможность управления шиной ISA - *Bus-Mastering* - со стороны адаптера. Это внешнее управление шиной опирается на контроллер DMA, выполняющий в данном случае функции арбитра шины. Для получения управления шиной внешний Bus-Master посылает запрос по линии DRQx (только для каналов 5-7) и, получив подтверждение DACKx, устанавливает сигнал MASTERS. Теперь шиной ISA управ­ляет он, но формально он не имеет права занимать шину больше чем на 15 мкс за сеанс. В противном случае нарушится регенерация памяти (позже собьется системное время, но при нарушении регенерации эти «мелочи» уже не важны). Интеллектуальный контроллер может выполнять более эффективные процедуры обмена, чем стандартный DMA, например:

Scatter Write — «разбросанная» запись в несколько блоков памяти.

Gather Read - чтение со сбором данных из нескольких блоков памяти.

Обмен нечетным количеством байт и (или) с нечетного адреса по 16-бит­ному каналу.

Управление шиной используют высокопроизводительные адаптеры SCSI и локальных сетей, а также интеллектуальные графические адаптеры. Однако архитектурой шины доступное им пространство памяти ограничено областью 16 Мбайт, что по нынешним меркам маловато. «Заботливые» операционные системы (например, Novell NetWare) для таких адаптеров позволяют под буферы резервировать область в пределах младших 16 Мбайт.

На шине EISA DMA-каналы могут работать в 8-, 16- и 32-битном режиме, они могут использовать все 32 разряда шины адреса — иметь доступ ко всей памяти компьютера. Каждый канал может программироваться на 1 из 4 типов цикла передачи:

Compatible — полностью совместим с ISA.

Type A — сокращенный на *25%* цикл: время одиночного цикла 875 нс, в блочном режиме время цикла 750 нс. Работает почти со всеми ISA-адап­терами с большей скоростью.

Type В - сокращенный на 50% цикл (750/500 нс на цикл), работает с большинством EISA-адаптеров и некоторыми ISA. Этот тип цикла возмо­жен только с памятью, непосредственно доступной контроллеру шины EISA (памятью на адаптерах EISA, а также системной в случае, если EISA является основной шиной системной платы). Если декодированный адрес памяти относится к 8/16-битной памяти ISA, то контроллер DMA EISA автоматически переводится в режим Compatible.

Type С (Burst Timing) — сокращенный на 87,5% цикл, ориентированный на пакетный режим передач. Работает со скоростными EISA-адаптерами и при обмене 32-битных устройств с 32-битной памятью позволяет разви­вать скорость обмена до 33 Мбайт/с.

В PCI-системах для обмена с устройствами системной платы (Fast ATA-2 или E-IDE-порты) возможно использование DMA Type F, при котором между соседними циклами интервал может не превышать 3 тактов шины (360 нс). Для разгрузки системной шины используется дополнительный 4-байтный буфер. Режим F может работать только в режиме одиночной передачи или по запросу и только с инкрементом (увеличением) адреса. На самой шине PCI адаптеры могут использовать режим прямого управления шиной, для чего имеется специальный протокол арбитража, который к контроллерам DMA отношения уже не имеет.

**Таблица 1. Стандартные каналы прямого доступа к памяти.**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер канала DMA# | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Стандартное назначение | XT | MRFR\* | - | FDD | HDD | Отсутствуют | | | |
| AT | - | SDLC\* | FDD | HDD\* | Каскад | - | - | - |
| Разрядность, байт | | 1 | | | | 2 с четного адреса | | | |
| Макс. размер блока | | 64 Кбайта | | | | 128 Кбайт, четный | | | |
| Граница блоков | | Кратна 1000h | | | | Кратна 2000h | | | |
| Регистр страниц | | 4 бит А16-А19 | | | | 7 бит А17-А23 | | | |
| Адреса регистров: | |  |  |  |  |  |  |  |  |
| страниц | | 087 | 083 | 081 | 082 | 08F | 08В | 089 | 087 |
| нач. адреса (W)  текущ. адреса (R) | | 000 | 002 | 004 | 006 | 0С0 | 0С4 | 0С8 | 0СЕ |
| нач. счетчика (W)  текущ. счетчика (R) | | 001 | 003 | 005 | 007 | 0С2 | 0С6 | 0СА | 0СЕ |

\*SDLC-адаптер устанавливается редко.

HDD-контроллер в AT DMA обычно не использует.

Канал 0 в XT используется для регенерации памяти (MRFR).

Канал 4 доступен только в PS/2 МСА.

## Программное управление контроллером ПДП

Программное управление контроллером ПДП осуществляется через порты ввода-вывода. Доступ к каждому регистру контроллера может быть осуществлен через свои порты ввода-вывода. Распределение адресов и описание внутренних регистров первого и второго контроллера ПДП приведено в таблице 2.

**Таблица 2.** **Регистры контроллера ПДП 8237А.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **8237#1** | | **8237#2** | | **R/W** | | **Назначение регистров** |
| 008h | | 0D0h | | W | | Регистр команд (Command Register)  Биты: 7=1 – активный уровень DACK – высокий  6=1 – активный уровень DRQ – высокий  5=1 – режим расширенной записи  4=1 – циклический приоритет  3=1 – укороченный цикл обмена  2=1 – запрет работы контроллера  1=1 – фиксация адреса 0 канала  0=1 – передача память-память (в PC не используется) |
| 008h | | 0D0h | | R | | Регистр состояния каналов (Status Register)  Биты 7..4 запросы каналов 0-3  Биты 3..0 завершение цикла каналов 0-3 |
| 009h | | 0D2h | | W | | Регистр запросов (Request Register)  Биты 7..3 – не используются  2=1 – установка/ =0 – сброс бита запроса  1..0 – выбор канала (00=0, 01=1, 10=2, 11=3) |
| 00Ah | | 0D4h | | W | | Регистр маски - Single Mask Bit Register  Биты 7..3 – не используются  2=1 – установка/ =0 – сброс бита маски  1..0 – выбор канала (00=0, 01=1, 10=2, 11=3) |
| 00Bh | | 0D6h | | W | | Регистр режима работы канала (Mode Register)  Биты 7..6 – режим передачи (00- по запросу, 01- одиночный, 10- блочный, 11- каскадирование)  5=1 – инкремент / =0 – декремент адреса  4=1 – разрешение автоинициализации  3..2 – тип передачи (00- холостой, 01-запись, 01- чтение, 11- не исп.)  1..0 – выбор канала (00=0, 01=1, 10=2, 11=3) |
| 00Ch | | 0D8h | | W | | Сброс триггера младшего/старшего байта – Clear Byte Pointer Flip/Flop |
| 00Dh | | 0DAh | | W | | Общий сброс 8237А - Master Clear (вывод любого байта в регистр вызывает сброс) |
| 00Eh | | 0DCh | | W | | Общий сброс масок всех каналов – Clear Mask Register (вывод любого байта в регистр вызывает сброс) |
| 00Fh | 0DEh | | W | | Регистр масок всех каналов - All Mask Register Bits  Биты 7..4 – не используются  3..0 – маски каналов 0-3 (0-канал разрешен, 1- маскирован) | |
| Регистры управления каналами второго контроллера | | | | | | |
| - | 0C0h, 0C4h, 0C8h, 0CCh | | W | | Запись начального адреса в регистр начального адреса (Base Address Register) и регистр текущего адреса канала (Current Address Register) 4,5,6,7 | |
| - | 0C0h, 0C4h, 0C8h, 0CCh | | R | | Чтение начального адреса из регистра начального адреса канала (Current Address Register) 4,5,6,7 | |
| - | 0C2h, 0C6h, 0CAh, 0CEh | | W | | Запись в регистр начального счетчика циклов (Base Word Count Register) и в регистр текущего счетчика циклов канала (Current Word Count Register) 4,5,6,7 | |
| - | 0C2h, 0C6h, 0CAh, 0CEh | | R | | Чтение текущего значения из регистра текущего счетчика циклов канала (Current Word Count Register) 4, 5, 6, 7 | |
| - | 089h, 08Bh, 08Ah, 08Fh | | W | | Задание номера страницы для канала 6,5,7,4 | |

Каналы 4 - 7 предназначены для обмена 16-разрядными словами. В связи с этим возникает ряд отличий в работе с этими каналами:

- бит 0 в данных, заносимых в регистры начального и текущего адреса, всегда подразумевается равным 0, поэтому через эти регистры передаются биты 1 - 16 полного 23-разрядного адреса (а не биты 0 - 15 полного 20-разрядного адреса, как это реализовано на ХТ - подобных ПЭВМ). По этой же причине в страничные регистры каналов 4 - 7 заносятся биты 17 - 23 полного адреса, а не биты 16 - 23, как это надо сделать при работе с каналами 0 - 3;

- поскольку передача осуществляется 16-разрядными словами, в регистры текущего и начального счетчика циклов заносится не число байт, а число слов, уменьшенное на единицу;

- размеры страниц памяти, в пределах которых возможен обмен в течение одной передачи, составляют 2000h байтов.

**Описание регистров**

**Регистр начального адреса (Base Address Register).** В этом регистре задается стартовый адрес ОЗУ, с которого начинается передача. Регистр содержит 16 разрядов и определяет адрес внутри заданной страницы памяти размером 64К. Задание номера страницы памяти осуществляется через специальные страничные регистры (Page Registers), поддерживаемые внешней логикой.

Каждый канал ПДП имеет свой регистр начального адреса и страничный регистр. Такое деление памяти на страницы не позволяет осуществить обмен с блоком памяти, находящимся на пересечении двух страниц. Каждая страница начинается с сегментного адреса, кратного 1000h (0, 1000h, 2000h, ..., 9000h).

**Регистр начального счетчика циклов (Base Word Count Register)**

В этом регистре задается начальное число циклов передачи для программируемого канала. Фактическое число передаваемых во время работы ПДП элементов данных на единицу превышает заданное число циклов, т. е. если Вы задаете 100 циклов передачи, а размер элемента будет равен 1 байту, то за сеанс обмена будет передан 101 байт информации.

**Регистр текущего адреса (Current Address Register)**

Начальное значение заносится в этот регистр одновременно с регистром начального адреса. В дальнейшем в ходе передачи значение текущего адреса автоматически увеличивается или уменьшается (конкретное направление изменения задается при программировании в регистре режима). Если разрешена автоинициализация, то после окончания передачи в регистр автоматически устанавливается значение из регистра начального адреса.

**Регистр текущего счетчика циклов (Current Word Count Register)**

Регистр содержит текущее значение счетчика циклов (число оставшихся циклов передачи). Отображаемое в нем число циклов всегда на единицу меньше числа еще не переданных элементов данных, так как изменение значения в этом регистре производится в конце цикла передачи, уже после фактической передачи элемента данных, а конец передачи фиксируется в момент переполнения счетчика (изменение его значения с 0 на 0FFFFh).

Каждый из четырех каналов ПДП имеет свой набор регистров, описанных выше. Кроме того, имеется следующий набор регистров, общих для всех каналов.

**Регистр режима (Mode Register)**

Данный регистр задает режимы работы своего канала контроллера.

**Регистр команд (Command Register).**

Этот 8-битный регистр управляет работой контроллера. Он программируется, когда контроллер находится в состоянии программирования и очищается командами сброса «Reset» и «Master Clear».

**Регистр состояния (Status Register)**

Регистр отражает текущее состояние запросов и передач по всем четырем каналам. Биты 0 - 3 устанавливаются в единицу после завершения передачи по каналам 0 - 3 (бит 0 - канал 0, бит 1 - канал 1 и т.д.), если не задан режим автоинициализации. Эти биты очищаются после команды сброса контроллера и после каждой операции считывания состояния из регистра состояния. Биты 4 – 7 указывают по какому из каналов 0 - 3 активен в текущий момент сигнал запроса на ПДП.

**Регистр масок (Mask Register)**

Каждый бит этого 4-битового регистра маскирует/демаскирует свой канал ПДП, при этом значение 1 маскирует канал, значение 0 демаскирует канал и разрешает прием сигнала запроса по этому каналу.

**Регистр запросов (Request Register)**

Сигнал запроса на ПДП (DREQ) может быть издан как обслуживаемым устройством, так и программно. Для программного издания сигнала запроса по одному из 4-х каналов ПДП необходимо установить соответствующий бит в 4-разрядном регистре запросов. Запрос на ПДП может быть отменен записью нулевого значения в соответствующий бит регистра. Бит запроса очищается автоматически при окончании передачи по данному каналу. Все биты запросов очищаются при сбросе контроллера. Для того, чтобы воспринимать программные запросы на ПДП, канал должен находится в режиме блоковой передачи.

#### Литература

1. Уокерли Дж. Архитектура и программирование микроЭВМ: в 2-х кн./ пер. с англ. – М.: Мир, 1984.
2. Ассемблер. Юров. В. – СПб.: Питер, 2001. – 624 с.:ил.
3. Ассемблер: практикум. Юров. В. - СПб.: Питер, 2001. – 400 с.:ил.