**ФЛАГИ.**

Флаги можно устанавливать в 1/0.

Флаги состояния:

* AF - вспомогательный флаг переносов, используется для адресной арифметики(БСД-коды) – перенос/заёмы.
* ZF – признак нуля. Устанавливается в единицу при нулевом результате последней операции
* SF – признак знака. Флажок SF = 1, если старший бит результата равен единице. Т. Е он повторяет значение старшего бита результата, который при использовании доп. кода определяет знак числа.
* TF – бит устанавливается программистом, если TF = 1, то при завершении каждой программы возникает прерывание работы микропроцессора. Обычно этот бит использует в режиме отладки для передачи управления подпрограмме, которая определяет текущее состояние программы (например, содержимое регистров).
* IF – флаг прерывания, признак разрешения прерывания. При ИФ = 1, микропроцессор реагирует на внешние программные прерывания по входу INTR. ІФ = 0 – прерыванія запрещаются. Значение этого флага не влияет на восприятие внешних немаскируемых прерываний по входу NMI или программных прерываний по команде INT. Признак можно установить командами sti = 1, cti = 0;
* DF – флаг направления. Исп. для обработки данных при выполнении строковых команд. Для указания элементов данных(байтов, слов и т.д.) в каждом блоке используются индексные регистры SI, DI. После обработкі очередного элемента, микропроцессор автоматически изменяет содержимое индексных регистров для выбора следующего элемента. При этом, если D = 0, то команды обрабатвыющие данными увеличивают содержимое индексных регистров, если же d = 1, то команды уменьшают содержимое индексных регистров.
* OF – признак переполнения, устанавливается в единицу в случае переполнения при операциях сложения/вычитания со знаковыми числами, тем самым указывает на потерю старшего бита резуьльта.

Флаги младшего байта регистров признаков устанавливаются арифметическими или логическими операциями микропроцессора. Также используются командами условных переходов для изменения порядка выполнения программы.

Флаги старшего байта(кроме переполнения) отражают состояние микропроцессора и оказывают влияние на выполнение программы.

PUSHF – записать в стек регистров флагов,  
POP ax – извлекаем из стека и заносим в ах  
PUSH ax  
POPF

CS:IP – адрес следующей команды

db – byte  
dw – word (2 байта)  
dd – double word(3 байта)  
dq – 4 байта  
dt – исп. Для BCD-кодов(10 байтов)

**СЕГМЕНТНАЯ АДРЕСАЦИЯ ПАМЯТИ.**

Пусть исполняемый модуль некоторой программы загружен в память компьютера. Комныд модуля в заданом порядке считываются в микропроцессор и выполняются. При этом они используют данные, которые выбирают из памяти и регистров микропроцессора. Поскольку для адресации памяти микропроцессор 8086 используют 16-битовые адрессные ригистры, это обеспечивает доступ к 65536 байтам(64кбайт) основной памяти, в то время общее адресное простравносвто состовляло 1мбайт.

Рассмотрим как можно адрессовать пам'ять, объемов 1мегабайт с помощью 16-битовых регистров микропроцессора. Для этого необходимо разделить всю адрессуемую память на сегменты, объемов 64кбайт. Причём принято, каждый сегмент может начинаться на границе параграфа(16 байт) от начала памяти.

Различные сегменты могут перекрываться или совпадать друг с другом. Начальный адресс каждого сегмента должен хранится в одном из сегментных регистров. Все эьти регистры 16-разрядные. Таким образом, стартовые адресса будут хранится в регитстрах, причем значения адресов без последнего 16-ричного нуля. Этии адреса называются базовыми адресами сегментов.

Для того, чтобы получить полного физического адреса начала сегмента, к содержимому регистра достаточно дописать 16-рияный ноль справа.

Физический адрес некоторого элемента в памяти(порядковый номер первого байта элемента от начала памяти) определяется суммой значения заданного в сегментном регистре со значением называемом смещением.

Смещение определяет порядковый номер первого байта элемента от начала сегмента и может хранится в одном из следующих регистров микропроцессора: IP - для кода; SI, DI, BX – данные; SP, BP – сегмент стека.

Сумірованіе просіходіт след. Образом: микропроцессор расширятет содрежимое сегментного регитсра (базовый адрес),добавляя к нему 4 млдшіх 16-ричных бита. И прибавляет к младшим 16 битам значение смещения. Получкенный 20-битовый наз. абсолютным адресом сегмента.

В командах асика можно использовать префикс замены, который позволяет переобпределять заданные по умолчанию сегменты для разных элементов. Значения смещений в адресных регистров можно увеличичвать или уменьшать, но при это миметь ввиду, что память имеет кольцевую организацию, т.е после смещения FFFFF следует 00000. Как правіло для каждой памяти выделяется свои сегменты. Основынх сегмента три: код, данные, стек.

**СПОСОБЫ АДРЕСАЦИИ.**

Большинство команд процессора содержит аргументы, которые принято называть операндами. Существующие способы задания адресов хранения операндов называются способами адресации. Способы адресации:

1. Непосредственная адресация
2. Прямая адресация
3. Регистровая адресация
4. Косвенная регистровая адресация
5. Относительная косвенная регистровая адресация
6. Базовая индексная адресация
7. Неявная адресация

ПРИМЕР 10.

Для матрицы, определенной в сегменте данных вычислить суммы элементов столбцов.

.model small  
.stack 100h  
.data  
n dw 4 ; размеры матрицы: число столбцов  
m dw 5 ; число строк  
x dw 1, 2, 3, 4 ; матрица  
 dw 5, 6, 7, 8  
 dw 9, 1, 2, 3   
 dw 4, 3, 2, 1  
 dw 0, 2, 7, 9  
lnstr dw 8 ;длина строки матрицы  
lnel dw 2 ;длина элемента матрицы  
s dw 4 dup(?) ;массив для сумм

.code  
begin:  
 mov ax, @data ;позиционируем регистр ds на сегмент данных  
 mov ds, ax  
 mov cx, n  
 xor bx, bx  
 l2:  
 push cx  
 xor si, si  
 xor ax, ax ;обнуление  
 mov cx, m ;организуем счетчик внутренних циклов  
 l1: ;по строкам  
 add ax, x[bx+si]  
 add si, lnstr ;переходим на следующую строку путем добавления 8  
 loop l1 ; уменьшает содержимое cx на 1 и если там нет 0, выполняется переход к указанной метке(в данном случае l1)  
 mov s[bx], ax  
 add bx, lnel  
 pop cx  
 loop l2

Mov ax, 4c00h ; завершает программу и передаёт управление ОС  
 int 21h

End begin

**ОРГАНИЗАЦИЯ СТЕКА.**

**ПРОСТЕЙШИЕ ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ СЕГМЕНТОВ.**

.model – задает модель памяти для ассемблерной программы.   
.data -   
.code -   
.stack -   
.data ? -   
.const -   
.fardata -   
.dosseg –

**ВЫПОЛНЕНИЕ ЛОГИЧЕСКИХ И СДВИГОВЫХ ОПЕРАЦИЙ.**

And  
or  
Xor  
Not

Эти операции побитовые.

**ОРГАНИЗАЦИЯ ЦИКЛОВ УСЛОВНЫХ И БЕЗУСЛОВНЫХ ПЕРЕХОДОВ.**

Пример:

.model  
.stack  
.data  
my\_string db “нажата клавиша ESC”, 10, 13, ‘$’

.code  
start:  
 mov ax, @data  
 mov ds, ax  
 mov ah, 1 ;функция ввода символа  
 int 21h  
 cmp al, 27 ;27 – код клавиши ESC  
 jne go\_to\_end  
 lea dx, my\_string  
 mov ah, 9 ;функция вывода строки  
 int 21h

go\_to\_end:  
 mov ah, 4ch  
 int 21h

end start

ПРИМЕР. ТРАНСПОНИРОВАНИЕ БИТОВОЙ МАТРИЦЫ.(будет в билетах)

.data  
massre db 8 dup(?) ;исходник (ресурс)  
masdst db 8 dup(?) ;массив приемник

.code

Mov cx, 8  
lea di, masdst

----------------------------------------------

Cycle1:  
 push cx  
 mov cx, 8  
 lea si, massrc  
 cycle:  
 rol BYTE PTR[si], 1  
 rcl al, 1  
 inc si  
 loop cycle  
 pop cx  
 mov [di], al  
 inc di  
loop cycle1

----------------------------------------------

**Работы со строками.**

Для цепочки-источника это регистр **esi/si.**  
Для цепочки-получателя это регистр **edi/di.**

Направление работы со строками осуществляется с помощью флажка df:  
df = 0 – inc, поставить в ноль – cld, df = 1 – dec, поставить в 1 - sld.

*Префиксы повторений.*

**Rep  
repe или repz  
repne или repnz**

*Пересылки данных.*

**Movs** приемник, источник  
**movsb  
movsw  
movsd**

*Сравнение цепочек.*

*CMPS*

*Сканирование цепочек.*

*Загрузка элемента из источника.*

*Сохранение элемента.*

Листинг 1. Пересылка строк командой movs.

;prg 11.1 asm  
.model small  
.stack 256  
.data  
source db “Тестирующая строка$” ;строка источник  
dest db 19 dup(“ ”) ;строка приемник  
.code

Main:  
 mov ax, @data  
 mov ds, ax  
 mov es, ax  
 cld ;обработка строки от начала к концу

Lea si, source  
 lea, di, dest  
 mov cx, 20

Rep movs dest, source ;пересылка строки ; movsb;  
 lea dx, dest  
 mov ah, 09h  
 int 21h  
 mov ax, 4c00h  
 int 21h  
end main

Листинг 3. Поиск символа в строке командой scas

model small  
.stack 256  
.data  
fnd db 0ah, 0dh, ‘Символ найден!$’

nochar db 0ah, 0dh, ‘Символ не найден.$’

string db ‘Поиск символа в этой строке.’, 10, 13, “$”

.code

ASSUME ds:@data, es:@data  
main:

Mov ax, @data  
 mov ds, ax  
 mov ah, 09h  
 lea dx, string  
 int 21h

Mov al, ‘a’

Cld  
 lea di, string  
 mov cx, 29  
 repne scasb  
 je found

failed:

mov ah, 09h  
 lea dx, nochar  
 int 21h  
 jmp exit  
found:

mov ah, 09h  
 lea dx, fnd  
 int 21h

;теперь, чтобы узнать место, где совпал элемент в строке, необходимо уменьшить значение в регистре di и вставить нужный обработчик  
dec di

Exit????

Листинг 2. Сравнение двух строк командой cmps

.model small  
.stack 256  
.data  
match db 0ah, 0dh, ‘Строки совпадают$’  
failed db 0ah, 0dh, ‘Строки не совпадают$’  
string1 db ‘0123456789’, 0ah, 0dh, ‘$’  
string2 db ‘0123456789’, ‘$’

.code

ASSUME ds:@data, es:@data

…

Mov ah, 09h  
lea dx, string1  
int 21h  
lea dx, string2  
int 21h  
cld  
lea si, string1  
lea di, string2  
mov cx, 10

Cycl:

Repe cmsb  
 jcxz equal ;cx = 0, типа строки равны))  
 jne not\_match ;если строки не равны  
 equal:  
 mov ah, 09h  
 lea dx, failed  
 int 21h  
 jmp exit  
not\_match:  
 mov ah, 09h  
 lea dx, failed  
 int 21h

Dec si  
 dec di

Inc si  
 inc di  
 jmp cycl  
exit:

Mov ax, 4c00h  
 int 21h

End main

Пример: Программа выводит на экран все ASCII-символы без исключения, используя прямой вывод на экран.

.model tiny  
.code  
.386  
org 100h

Start:

Mov ax, 003h  
 int 10h ;видеорежим 3(очистка экрана)  
 cld ;обработка строк в прямом направлении  
 mov eax, 1F201F00h ;первый символ 00 с атрибутом 1Fh,  
 ;затем пробел (20h) с атрибутом 1Fh  
 mov bx, 0F20h ;пробел с атрибутом 0Fh  
 mov cx, 255  
 mov di, offset ctable

Cloop:  
 stosd ;записать пробел и символ в таблицу ctable  
 ine al ;AL содержит следующий символ  
 test cx, 0Fh ;если CX не кратен 16, то…  
 jnz continue\_loop  
 push cx ;сохранить значение счетчика  
 mov cx, 80-32 ;число оставшихся до конца строки символов  
 xchg ax, bx   
 rep stows ;заполнить остаток строки

;пробелами с атрибутом0F

Xchg bx, ax  
 pop cx  
continue\_loop:

Loop cloop

Stosd ;записать последний 256 символ и пробел  
 mov ax, 0b800h  
 mov es, ax  
 xor di, di  
 mov si, offset ctable  
 mov cx, 15\*80+32  
 rep movsw  
 ret  
ctable:

End start

Программа перевода число из одной сс в другю сс:

.model small  
.stack 100h  
.code  
start: Mov ax, 12h  
 mov bx, 2 ;bx – новое основание сс(2<=bx<=16)  
 call my\_convert  
 mov ah, 4c00h  
 int 21h

My\_convert PROC

Push cx, dx  
 sub cx, cx

Again:  
 sub dx, dx  
 div bx  
 inc cx  
 push dx  
 cmp ax, 0  
 jne again  
loop\_output:  
 pop dx  
 add dx, 30h  
 cmp dx, 39h  
 jle no\_more\_9  
 add dx, 7  
no\_more\_9:  
 mov, ah, 2  
 int 21h

Loop loop\_output

pop dx, cx  
 ret  
my\_convert ENDP

Пример. В процедуру передается строковая запись числа. Требуется перевести ее из строки в число и возвратить число на регистре.

.model small  
.stack 100h  
.data  
num db 5 dup(?) ;выделяем память под строку  
.code  
start:

Mov ax, @data  
 mov ds, ax  
 mov ah, 3fh  
 ;функция ввода строки   
 xor bx, bx ;дескриптор stdin равен нулю  
 mov cx, 4 ;максимальное кол-во символов  
 lea dx, num ;адрес буфера для строки  
 int 21h ;в ах будет кол-во введенных знаков  
 lea di, num ;передаем адрес строки  
 call stri ;вызываем процедуру перевода

Mov ax, 4c00h ;выход из программы  
 int 21h

Stri proc near ;процедура перевода строки в число  
 ;адрес строки передается в di  
 ;длина строки в ах  
 ;число возвращается в ах  
 push di cx bx dx ; сохраняем содержимое регистров  
 mov cx, ax ;кол-во циклов равно длине строки  
 mov bl, 10 ;предполагаем десятичное число  
 xor ax, ax ;здесь будет результат  
 xor bh, bh  
 xor dh, dh

Next\_char:  
 mul bl ;ax=al\*bl  
 mov dl, [di] ;берем символ из строки  
 sub dl, 30h ;преобразуем его в цифру  
 add ax, dx ;число формируем в регистре ах  
 inc di ;переходим к следующему символу  
 loop next\_char

Exit:  
 pop dx bx cx di ; восстанавливаем содержимое регистров   
 ret

Stri endp

End start

ПРИМЕР:

Процедура добавляет значение, находящемся в регистре АХ, к неупорядоченному списку(массиву) в доп. сегменте(при условии, что такого элемента в списке нет). Начальный адрес списка берется из регистра DI. По возвращению из процедуры значения регистров DI и AX не изменяются.

ADD\_TO\_UL proc

CLD ;DF=0 для сканирования слева  
 push di cx  
 mov cx, es:[di]  
 add di, 2  
 repne scasw ;сравнение АХ со значением ES:[DI]  
 pop cx  
 jne ADD\_IT  
 pop di  
 ret

ADD\_IT:

Stosw ;записываем АХ по адресу ES:[DI]  
 pop di  
 inc word ptr es:[di]  
 ret  
ADD\_TO\_UL endp

ПРИМЕР:

Процедура удаляет значение, находящемся в регистре АХ, из неупорядоченного списка(массиву) в доп. сегменте(при условии, что такого элемента в списке есть). Начальный адрес списка берется из регистра DI. По возвращению из процедуры значения регистров DI и AX не изменяются.

DEL\_UL proc  
 cld  
 push bx di  
 mov cx, es:[di]  
 add di, 2  
 repne scasw  
 je DELETE  
 pop di bx  
 ret

;Следующие команды удаляют элемент из списка по агоритму:

;1) если элемент находится в конце списка, удалить его путем уменьшения счетчика элементов на 1  
;2) в противном случае удалить его путем сдвига всех следующих за ним элементов на одну позицию влево

DELETE:

Jcxz dec\_cnt  
NEXT\_EL:  
 mov bx, es:[di]  
 mov es:[di-2], bx  
 add di, 2  
 LOOP NEXT\_EL

DEC\_CNT:

Pop di  
 dec word ptr es:[di]  
 pop bx  
 ret  
DEL\_UL endp

ПРИМЕР:

Процедура находит MINMAX, находящемся в регистре АХ, к неупорядоченному списку(массиву) в доп. сегменте(при условии, что такого элемента в списке нет). Начальный адрес списка берется из регистра DI. По возвращению из процедуры значения регистров DI и AX не изменяются.

MINMAX proc  
 push cx di  
 mov cx, es:[di]  
 dec cx  
 push cx  
 mov bx, es:[di+2]  
 mov ax, bx

Add di, 4  
 push di

CHKMIN:

Cmp es:[di], bx ;сравнить следующий  
 jae NOMIN ;найден новый мінімум?  
 mov bx, es:[di] ;да, поместить его в ВХ  
NOMIN:  
 add di, 2 ;нет, перейти к следующему  
 loop CHKMIN ;проверить весь список

Pop di, cx

CHKMAX:

Cmp es:[di], ax  
 jbe NOMAX  
 mov ax, es:[di]  
NOMAX:  
 add di, 2  
 loop CHKMAX

Pop di cx  
 ret

**ВВОД-ВЫВОД С КЛАВИАТУРЫ.**

Размер буфера клавиатуры – 32 байта.

Буфер начинается с адреса 40:1Еh и заканчивается 40:3Eh.

Для доступа к буфера используется два указателся: указатель “головы” по адресу 40:10h и “хвоста” по адресу 40:1Сh. В этих указателях хранятся смещения от адреса “головы”. При каждом помещении символа в буфер указатель хвоста наращивается. И так как адресация буфера реализована циклически, то при непрерывном нажатии клавиш буфер заполнится, запись в буфер блокируется и подается звуковой сигнал.

Ввод данных с клавиатуры м/б реализован с помощью функции DOS/BIOS.

Пример: Дан массив ASCIIZ строк. Записать в отдельный массив их длины. Найти самую длинную строку, скопировать ее в буфер, и если в ней присутствуют элемент “!”, преобразовать строку к верхнему регистру.

.model small  
.stack 100h  
.data

StrQnty equ 6 ; кол-во строк

CopyLength equ 80 ; максимальная длина строки

String\_copy db CopyLength dup(?) ;буфер для строки

String1 db “Assembler is cool!!!”, 0  
String2 db “C++ is cool!!!”, 0  
String3 db “KSIS-SILA!!!”, 0  
String4 db “It is a good day to die…”, 0  
String5 db “We are the champions.”, 0  
String6 db “Microsoft Must die!”, 0

;массив смещений соответствующих строк

StrSArray dw offset string1, offset string2, offset string3  
 dw offset string4, offset string5, offset string6

LenArray dw StrQnty dup(?) ;массив длин строк

.code  
start:

Mov ax, @data  
 mov ds, ax  
 mov es, ax ; загрузка регистров ES=DS

Cld ; установка флага DF в 0  
 mov cx, StrQnry ; кол-во строк  
 mov bx, offset LenArray ;смещение массива для строк  
 mov si, offset StrSArray ; смещение массива смещений строк

Calc\_lengths: ; вычисление длины строки

Jcxz fin\_calc\_lengths ; если строк нет, выход из цикла  
 mov di, [si] ; загрузка адреса строки на DI  
 xor al, al ; al = 0 – поиск 0  
 mov dx, cx ; сохранение значения CX на DX  
 mov cx, -1 ; длина строки неизвестна

Loop\_strlen:  
 inc cx  
 scasb ; сравниваем содержимое AL с байтом строки  
 jne loop\_strlen   
 mov [bx], cx ; запись длины строки в массив  
 mov cx, dx ; восстановление сохраненного значения  
 add si, 2 ; переход на следующую строку  
 add bx, 2  
 loop calc\_lengths

Fin\_calc\_lengths:

; дальше будем искать строку максимальной длины

Xor ax, ax ; на AX будет максимальная длина  
 xor bx, bx ; на BX – смещение длины строки  
 mov cx, StrQnty   
 mov di, offset LenArray

Find\_maxlen: ; поиск строки максимальной длины  
 jcxz fin\_find\_maxlen ; если строк нет, выход из цикла  
 scasw ; сравнение AX со значением в [DI],  
 jnb find\_maxlen ; если не меньше, продолжить,  
 mov bx, di ; иначе – загрузка на AX и BX  
 sub bx, 2 ; новых значений  
 mov ax, [bx]  
 loop find\_maxlen

Fin\_find\_maxlen:

; переход от смещения длины в массиве длин строки (LenArray)  
; к смещению строки в массиве смещений строк (StrSArray)  
 add bx, offsetStrSArray-offset LenArray

; сейчас в bx смещение смещения строки с максимальной длиной  
 mov cx, ax ; на CX – длина строки  
 inc cx  
 mov si, [bx] ; смещение строки с максимальной длиной  
 mox di, offset string\_copy  
 mov bx, di  
 rep movsb ; копирование строки

mov di, bx ; восстанавливаем di  
 mov cx, ax  
 mov al, ‘!’

repnz scasb ; поиск в строке знака '!'  
 jnz exit ; если элемент не найден, выход

mov si, bx  
 mov di, bx

uppercase: ; переход к верхнему регистру  
 lodsb ; загружает байт из ds:si в al  
 test al, al ; если AL = 0, достигнут конец строки   
 jz end\_uppercase ; выход  
 cmp al, ‘a’  
 jb continue\_uppercase  
 cmp al, ‘z’ ; если это строчная буква латинскогго алфавита  
 ja continue\_uppercase  
 add al, ‘A’-‘a’ ; преобразовать в заглавную

continue\_uppercase:  
 stosb ; записать символ на место es:di  
 jmp short uppercase

end\_uppercase:

exit:

mov ah, 4c00h ; завершение программы  
 int 21h

end start

**ПОВТОРЯЮЩИЕСЯ БЛОКИ И МАКРОСЫ.**

REPT ВЫРАЖЕНИЕ  
 КОМАНДЫ\_ИЛИ\_ДИРЕКТИВЫ

ENDM

REPT 3 ; три раза DB 100  
 DB 100

ENDM

Пример:

.model small  
.stack 100h  
.data

My\_label LABEL WORD

My\_value = 10 ; директива = подобна EQU

REPT 5 ; строится массив вида DW 10, 20, 30, 40, 50  
 DW my\_value  
 my\_value = my\_value + 10

ENDM ; конец повторяющегося блока

.code  
start:  
 mov ax, @data  
 mov ds, ax  
 xor ax, ax  
 xor bx, bx

REPT 5 ; блок повторяется 5 раз

Add ax, my\_label[bx] ;суммирование элементов массива  
 inc bx ; инкремент bx 2 раза для выбора следующего элемента массива  
 inc bx

ENDM ; конец повторяющегося блока

Mov ah, 4ch  
 int 21h

End start

Діректівы “=” подобна EQU. Главное отличие – метку в EQU нельзя переопределять, а в “=” - можно.

Макросы:

1. Макроопределение – набор комнад, содержащий набор каких-либо действий или алгоритмов.
2. Макрокоманда – вызов макроса
3. Макрорасширение(макроподстановка, макровставка) – вставка с заменой формальных параметров на фактические.

Имя\_макросa MACRO [формальные параметры]

Тело макроса

ENDM

Пример: Вычислить значение факториала.

.model small  
.stack 100h  
.code  
 My\_mac MACRO num ;; начало макро  
 mov cx, num ;; значение num в регистре cx  
 mov ax, 1

My\_loop: ;; в цикле вычисляется факториал num

Jcxz \_end\_ ;; если в cx 0, переход к метке  
 mul cx  
 loop my\_loop

\_end\_: ENDM

Start:  
 my\_mac 5 ;; вызов макро с параметром 5

; в регистре ах будет вычисленное значение  
 mov ah, 4ch  
 int 21h

End start

Если вызывать макрос, который имеет внутри себя какие-либо метки, несколько раз в программе, то могут происходить ошибки переопределения меток. Чтобы их избежать требуется использовать ключевое слово LOCAL:

My\_mac MACRO num  
LOCAL my\_loop, \_end\_

…

ENDM

В программе некоторые комментарии записаны с помощью двух “;;”. Если использовать один “;”, то при каждом вызове макроса такие комментарии вставляются вместе с макросом. При двух “;;” – комменты вставляются один раз в определение макроса. Таким образом экономится память.

PURGE ИМЯ\_МАКРОСА – удаляет макрос. Дальнейший вызов будет приводить к ошибке.

ЕЩЁ ПРИМЕРЫ:

Message MACRO strn ; макрос для вывода строки на экран  
 lea dx, strn ; адрес параметра strn на dx  
 mov ah, 9 ; функция вывода строки  
 int 21h

ENDM

Макрос вывода на экран строки string начиная с позиции с координатами x, y.

PRINT MACRO x, y, string ;; формальные параметры  
 push ax bx dx ;; сохраняем регистры в стеке  
 mov ah, 2 ;; функция установки курсора   
 xor bh, bh ;; номер видеостраницы(нулевая)  
 mov dh, y ;;строка  
 mov dl, x ;; столбец  
 int 10h  
 mov ah, 9 ;; функция вывода строки  
 mox dx, offset string ;; адрес строки в dx  
 int 21h  
 pop dx bx ax ;; восстанавливаем регистры

ENDM

; вызов макроса:

PRINT Xpos, Ypos, message

**СТАНДАРТНЫЕ ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ СЕГМЕНТОВ.**

SEGMENT

ENDS

Seg\_name SEGMENT align combine use ‘class’

….

Seg\_name ENDS

Align – выравнивание  
combine – объединение

BYTE - разрешено использовать любой стартовый адрес;

WORD – стартовый адрес должен находиться на границе слова;

DWORD – стартовый адрес должен находиться на границе двойного слова;

PARA – стартовый адрес должен находиться на границе параграфа(16 байт);

PAGE – стартовый адрес должен находиться на границе страницы(256 байт);

COMMON – сегменты могут перекрывать друг друга, размер объединенного сегмента равен размеру наибольшего из объединяемых сегментов;

PRIVATE – сегменты не будут объединяться с другими сегментами;

PUBLIC – сегменты объединяются в один сегмент, размер объединенного сегмента равен сумме размеров объединяемых сегментов;

STACK – сегменты объединяются в один, в построенном EXE-файле регистр SS адресует начало объединенного сегмента, а регистр SP – байт за последним словом объединенного сегмента.

Параметр use (использование) предназначен для процессоров, начиная с 80386.

Параметр class (класс) используется компоновщиком для определения последовательности расположения сегментов в памяти.

Директива ASSUME сообщает ассемблеру о соответствии между сегментными регистрами и сегментами программы.

Директива GROUP позволяет объединять несколько сегментов в одну группу. В итоге все сегменты группы можно адресовать с помощью одного сегмента.

Отметим ситуации в которых следует іспользовать простейшіе діректівы определенія сегментов:

1. Если ассемблерные модули объединяется с языками высокого уровня, то почти во всех случаях следует использовать простейшие директивы определения сегментов. Это позволит подерживать единообразный интерфейс, так как языки высокого уровня автоматически создают сегменты, с которыми работает программа.
2. Простейшие директивы определения сегментов следует использовать при написании малых или средних ассемблерных программ, поскольку они проще для понимания и более читабельны.
3. При написании больших многомодульных программ использующих смешанные модели памяти следует применять стандартные директивы определения сегментов, так как в этом случае программисту предоставляются все возможности по управлению, созданием и использованием сегментов.

**МУЛЬТИМОДУЛЬНЫЕ ПРОГРАММЫ.**

PUBLIC [язык] метка[, [язык], метка]…

Эта директива определяет метки программы, которые будут доступны в других модулях. Например:

.DATA  
PUBLIC my\_equ, my\_var

My\_equ EQU 25

My\_var DD 12345678h

.CODE

PUBLIC my\_proc  
my\_proc PROC

….

Call my\_proc

….

EXTRN определение [, определение…]

[язык] метка: тип [количество повторений]

Язык: С, Pascal, NoLanguge

Тип: ABS, BYTE, DATAPTR, DWORD, FAR, PROC, WORD

.data

EXTRN my\_equ: ABS, my\_var: DWORD

.code

EXTRN my\_proc: PROC

Main\_proc PROC

……..

Call my\_proc

……..

Здесь задаются метки, которые определены в этом и других модулях, котоые также могут использоваться в других модулях:

GLOBAL определение [, определение…]

.data

GLOBAL my\_out: DATAPTR, my\_in: WORD

My\_in DW 10

.code

GLOBAL my\_out\_code: NEAR, my\_in\_code: FAR

My\_in\_code PROC FAR

………

Call my\_out\_code

По умолчанию язык Ассемблера не различает нижние и верхние регистры. В процессе ассмеблирования все внешние метки с атрибутом PUBLIC, EXTRN, GLOBAL преобразуются к верхнему регистру. Для того чтобы сохранить разницу между строчными и прописными буквами при компиляции требуется использовать опции /mx – для внешних переменных, /ml – для всех переменных.

Пример:

;Этот модуль в файле F1.asm

.model small  
.stack 100h  
.data

number dw 47 ; преобразуемое число  
base dw 16 ; Основание системы счисления

PUBLIC number, base

Str\_rez db ‘Результат преобразования: $’

.code

EXTRN mess: PROC, my\_convert: PROC

Start:  
my\_proc PROC ;Главная программа тоже оформлена как процедура  
 mov ax, @data  
 mov ds, ax  
 lea dx, str\_rez ;Передача параметра в mess через регистр dx  
 call mess ;вывод сообщения str\_rez  
 call my\_convert ;преобразование числа 47 в число с другой сс  
 mov ah, 4ch  
 int 21h

My\_proc ENDP

; Этот модуль описан в файле F2.asm

.model  
.code  
PUBLIC mess ;процедура mess с помощью функции 9 прерывания 21h  
 ;выводит сообщение с адресом в ds:dx.  
mess PROC

Mov ah, 9 ;вывод сообщения  
 int 21h  
 ret  
mess ENDP

END

; Этот модуль описан в файле F3.asm

.model small  
.data  
EXTRN number: WORD, base: WORD  
.code

;процедура my\_convert преобразует число number в число в системе счисления base, положим 2<=base<=16

PUBLIC my\_convert

My\_convert PROC  
 push ax cx dx  
 mov ax, @data  
 mov ds, ax  
 mov ax, number ;число 47 в ах  
 sub cx, cx ;будем подсчитывать цифры

Again:  
 sub dx, dx  
 div base ;делим на основание системы счисления  
 ;dx:ax/base => частное в ах, остаток в dx  
 inc cx ;сосчитали цифру  
 push dx ;заносим ее в стек  
 cmp ax, 0 ;частное = 0?  
 jne again ;нет, переходим к again

Loop output: ;цикл вывода числа на экран  
 pop dx ;извлекаем остаток из стека(цифра)  
 add dx, 30h ;получаем ее ASCII-код  
 cmp dx, 39h ;проверка получения цифры больше 9  
 jle no\_more\_9 ;если <=9, переход к no\_more\_9  
 add dx, 7h ;получение ASCII-кода цифр А, В, …, Е, F

No\_more\_9:  
 mov ah, 2 ;вывод символа, код которого в dl  
 int 21h

Loop loop\_output

Pop dx cx ax  
 ret

My\_convert PROC

END

В итоге получим:  
F1.obj, F2.obj, F3.obj  
TLINK F1 F2 F3 <ВВОД>  
F1.exe

Пример: Программа демонстрирует работу с окнами в текстовом режиме.  
.model small  
.stack 100h  
.data  
txt db ‘TEXT’

in equ $-txt

npage db 0

row db 12

col db 20

strg db 20,1Eh, 20,1Eh, 20,1Eh, 20,1Eh  
 db ‘C’,38h, ‘T’,38h, ‘P’,38h, ‘O’,38h, ‘K’,38h, ‘A’,38h  
 db 20,1Eh, 20,1Eh, 20,1Eh, 20,1Eh

len equ $-strg

num equ len/2

.code  
start:  
 mov ax, @data  
 mov ds, ax  
 mov es, ax  
 mov ax, 0600h ;очистка экрана  
 mov bh, 17h ;атрибут экрана  
 mov cx, 0000h ;координаты очищаемого экрана от 00,00  
 mov dx, 184fh ;до 24,79 – весь экран  
 int 10h  
 mov ax, 0600h ;очистка второго экрана с координатами  
 mov bh, 20h ;атрибут окна  
 mov cx, 0A1Eh ;координаты очищаемого экрана от 10,30  
 mov dx, 0E31h ;до 14,49  
 int 10h  
 mov ah, 13h ;функция вывода на экран  
 mov al, 0 ;использовать атрибут и не перемещать курсор  
 mov bh, npage ;страница  
 mov bl, 07h ;атрибут  
 mov cx, word ptr in ;длина строки  
 mov dh, 11 ;координаты на экране  
 mov dl, 37 ;dh – Y, dl – X  
 lea bp, txt ;адрес строки  
 int 10h

Mov ah, 13h ;функция вывода на экран  
 mov al, 3 ;в строке коды знаков и их атрибуты  
 mov bh, npage ;страница  
 mov cx, num ;длина строки  
 mov dh, 13 ;координаты строки на экране  
 mov dl, 33 ;dh – Y, dl - X  
 lea bp, strg ;адрес строки  
 int 10h

Mov ah, 4ch  
 int 21h

end start

**СТРУКТУРА .EXE И .COM ПРОГРАММ.**

Размер программы .EXE может быть любой, в то время как для .COM максимально 64К. Размер .сом всегда меньше аналогичного .ехе файла, потому что в .сом есть заголовок.

В .сом программе стек автоматически генерируется в конце сегмента, а в .ехе его нужно объявлять самому.

Сегмент данных в .ехе обычно определяется, а в регистре DS инициализируется началом этого сегмента. В .сом программе только один сегмент.

В .ехе программе необходимо инициализовать регистр DS, а в .сом программе сегмент стека и сегмент данных – это один сегмент => этого делать не нужно.

Когда программа начинает выполняться, то все регистры хранят адрес PSP. Так как адресация начинается со смещения 100h от начала PSP и программа загрузчик заносит в IP 100h, то нужно указывать директиву ORG 100h.

Пример на сравнение чисел: В сегменте данных определены числа в формате двойного слова. С клавиатуры вводится число и программа определяет, имееться ли данное число в сегменте данных

Dat segment

C16 dw 10h ;число в 16 сс  
str1 db 10, 13, “Введите число для поиска:”, $  
str2 db 10, 13, “Число найдено$”  
str3 db 10, 13, “Заданного числа нет в сегменте данных”, 10, 13, ‘$’  
buf db 10, ?, 10 dup(?) ;буфер для числа

mat dd 256h, 1234567h, 99999999h, 0ABCDEFh, 1A2B3Ch ;числа в сегменте  
endmat dw ? ;метка конца чисел

dat ends

stk segment stack ‘stack’  
dw 100h dup(?)  
stk ends

cod segment  
 ASSUME ds:dat, ss:stk, cs:cod  
displ macro string ;макрос вывода строки  
 mov ah, 09h  
 lea, dx, string  
 int 21h  
endm

start:

mov ax, dat ;инициализация сегментного регистра dx  
 push ax  
 pop ds  
 mov ah, 0 ;очистка экрана  
 mov al, 3  
 int 10h

Disp str1 ;вывод строки приглашения

mov ah, 0ah ;ввод строки с числом  
 lea dx, buf ;буфер ввода  
 int 21h

mov cx, offset endmat ;определение кол-ва двойных слов в  
 sub cx, offset mat ;сегменте данных  
 shr cx, 2 ;соотв. Делению на 4(слова по 4 байта)  
 cmp cx, 0  
 je back1

call convert ;функция перевода строки в число  
 ;размещение в dx:ax  
 xor bx, bx

met1:  
 cmp ax, word ptr mat[bx] ;сравнение младшей части числа  
 jne met3

met2:  
 cmp dx, word ptr mat[bx+2] ;сравнение старшей части числа  
 je back2

met3:  
 add bx, 4 ;переход к следующему числу  
 loop met1

back1:  
 dicpl str3 ;вывод результата поиска  
 jmp back3

back2:  
 dicpl str2 ;вывод результата поиска

back3:  
 mov ah, 01h ;задержка  
 int 21h  
 mov ah, 4ch ;завершение программы  
 int 21h

convert proc ;Функция перевода строки в двойное слово помещаемое в dx:ax  
 push bx, cx, si  
 xor cx, cx  
 mov cl, buf+1 ;в cl помешаем кол-во символов в строке  
 xor ax, ax  
 xor dx, dx  
 mov si, 2

c1:  
 mov bx, dx  
 shl dx, 4 ;старшая часть числа умножается на 16  
 push dx  
 mul c16 ;младшая часть числа умножается на 16  
 mov bx, dx  
 pop dx  
 add dx, bx  
 xor bx, bx  
 mov bl, byte ptr [buf+si] ;в bl очередной символ  
 cmp bl, 61h (‘a’) ;если это буква нижнего регистра  
 jge c2  
 cmp bl, 41h (‘A’) ;если это буква верхнего регистра  
 jge c3

sub bx, 30h ;если это цифра  
 jmp c4

c2: ;преобразуем и накапливаем в число  
 sub bx, 57h  
 jmp с4

c3:  
 sub bx, 37h

c4:  
 add ax, bx  
 inc si ;выбираем следующий символ из буфера  
 loop c1  
 pop si cx bx  
 ret

convert endp

cod ends

end start

Пример(фрагмент программы) на умножение длинных чисел(в столбик как в школе AB\*CD):

OPR1 DW ?, ?

OPR2 DW ?, ?

REZ DW 0, 0, 0, 0

;-----------------------

Mov ax, OPR1 ;B  
mul OPR1, ;B\*D  
mov REZ, ax ;мл. часть BD  
mov REZ+2, dx ;ст. часть BD

;

Mov ax, OPR1 ;B  
mul OPR2+2 ;B\*C  
add REZ+2, ax ;BC  
adc REZ+4, dx ;adc – потому что учитываем перенос

;

Mov ax, OPR1+2 ;A  
mul OPR2 ;A\*D  
add REZ+2, ax ;AD  
adc REZ+4, dx

;

Mov ax, OPR1+2 ;A  
mul OPR2+2 ;A\*C  
add REZ+4, ax ;AC  
adc REZ+6, dx

Пример на вывод мтарицы: В процедуру передается адрес матрицы, размеры матрицы. Вывести матрицу на экран. Числа в матрице не больше 99.

.model small  
.stack 100h  
.data

Matr db 10, 20, 30, 40 ;массив  
 db 5, 26, 67, 2  
 db 12, 3, 42, 23  
i dw 4 ;размеры матрицы  
j dw 3

.code  
start:  
 mov ax, @data ;устанавливаем адрес сегмента данных  
 mov ds, ax  
 lea, dx, matr ;помещаем в стек параметры  
 push dx ;первый параметр – адрес матрицы  
 mov ax, i ;потом размеры матрицы  
 push ax  
 mov ax, j  
 push ax  
 call print ;вызываем процедуру  
 add sp, 6 ;освободим стек от параметров  
 mov ax, 4c00h ;завершение программы  
 int 21h

Print proc near  
 push bp ;сохраняем bp в стеке  
 mov bp, sp ;устанавливаем bp на вершину стека  
 mov ax, [bp+4] ;читаем размеры матрицы  
 mov bx, [bp+6]  
 mov dx, [bp+8] ;читаем адрес матрицы  
 mul bl ;определяем кол-во элементов и организуем цикл  
 mov cx, ax  
 mov bx, [bp+6]

Next:  
 mov al, [di] ;берем число из матрицы  
 xor ah, ah  
 mov dl, 10 ;перевод число в строку знаков  
 div dl  
 add ax, 3030h  
 mov dx, ax ;вывод строки-числа  
 mov ah, 02h  
 int 21h  
 xchg dl, dh  
 int 21h  
 mov dl, ‘ ’ ;вывод пробела между числами  
 int 21h  
 inc di ;переходим на следующее число  
 inc bh  
 cmp bh, bl ;переход на новую строку?  
 jne nstr  
 mov dl, 0ah ;да, переход на новую строку  
 int 21h  
 mov dl, 0dh  
 int 21h  
 xor bh, bh

Nstr:  
 loop next ;следующий цикл  
 pop bp ;восстанавливаем bp

print endp

ret

end start

**ИНТЕРФЕЙС ASSEMBLER НА С/С++**

Существует два способа:

* Вставки
* Использование внешних процедур(функций)

Преимущества второго способа: написание и отладку программы можно производить независимо, отлаженные модули можно использовать в других проектах, облегчается модификация и сопровождение программ.

ИСПОЛЬЗОВАНИЕ ВСТРОЕННОГО АССЕМБЛЕРА.

Объявление: *asm …*

Для встроенного ассемблера разрешается использовать строковые команды, машинные команды, команды перехода(передачи управления), директивы описания/определения данных(db, extrn…).

Нельзя использовать директивы такие как Assume, Proc, Segment, ORG, Dgroup, \_STACK, \_DATA, \_TEXT, ENDP, ENDS.

В ассемблерных вставках можно ссылаться на переменные С/С++.

**ВЫЗОВ ФУНКЦИИ НА С ИЗ ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА**

.model small  
.data

Str1 db “Минск”, 0  
str2 db “Минск”, 0  
str3 db “Киев”, 0

.code

Public C main  
EXTRN C fun ; near  
main PROC near

Mov ax, DGROUP  
 mov ds, ax

Lea ax, str2  
 push ax  
 lea ax, str1  
 push ax  
 call fun  
 add sp, 4  
 mov dx, ax  
 mov ah, 9  
 int 21h  
 lea ax, str3  
 push ax  
 lea ax, str1  
 push ax  
 call fun  
 add sp, 4  
 mov dx, ax  
 mov ah, 9  
 int 21h  
 mov ah, 4ch  
 int 21h

Main ENDP

End

//Вызываемая программа на С  
extern “C” char \*fun(char\* str1, char\* str2)  
{  
 char\* str;  
 for(; \*str1 == \*str2; str1++, str2++)  
 if(\*str1 == 0)  
 return (char\*) ”\x0D\x0AСтроки равны$”;  
 return (char\*) “\x0D\x0AСтроки равны$”;  
}

**КОМАНДНАЯ СТРОКА.**

ТУТ ПРИМЕР ДОЛЖЕН БЫТЬ

Пример. Программа анализирует содержимое командной строки. Выводит кол-во символов командной строки, а также кол-во параметров командной строки.

.model tiny  
.code  
org 100h

Start:  
 mov ah, 9  
 lea dx, msg1  
 int 21h

Mov ah, 40h ;выводим командную строку  
 mov bx, 1 ;на stdout  
 mov cl, byte ptr ds:80h ;длина строки  
 xor ch, ch  
 mov dx, 81h ;смещение начала строки  
 int 21h  
 mov ah, 9

Lea dx, msg2  
 int 21h

Call paramcount  
 push ax  
 call printint ;выводим на экран  
 pop bx ;очищаем стек

Mov ah, 9  
 lea dx, msg3  
 int 21h

Mov al, byte ptr ds:80h ;выводим длину строки  
 xor ah, ah  
 push ax ;число в стеке  
 call printint ;выводим на экран   
 pop bx ;очищаем стек  
 int 20h

;Процедура вычисляет и возвращает в регистре ах количество аргументов в ;командной строке

Proc paramcount near  
 push si  
 xor bx, bx ;счетчик аргументов  
 mov cl, byte ptr ds:80h ;длина строки аргументов  
 xor ch, ch  
 mov si, 81h ;смещение строки аргументов  
 cld

Skip\_separator:

Jcxz no\_chars ;cx>0?  
 dec cx

Lodsb ;загрузка в al очередного символа  
 cmp a, ‘ ’ ;это пробел?  
 je skip\_separator ;делаем пропуск  
 cmp al, ‘/’ ;это ‘/’?  
 inc bx ;еще один аргумент обнаружен

Skip\_argument:

Jcxz no\_chars ;все символы просмотрели?  
 dec cx ;похоже, что нет  
 lodsb ;копируем в al очередной символ  
 cmp al, ‘ ’ ;это пробел?  
 je skip\_separator  
 cmp al, ‘/’ ;это ‘/’?  
 je skip\_separator   
 jmp skip\_argument ;повторяем цикл

No\_chars:

Mov ax, bx ;возвращаем в ах кол-во аргументов  
 pop si ;восстанавливаем si  
 ret ;и выходим

Endp

;Процедура преобразует число X в строку знаков и посимвольно выводит ее ;на экран

Proc printint near

Arg x:word ;число х в стеке  
 mov ax, x  
 or ax, ax ;устанавливает флаг S если число < 0  
 jns l1 ;отрицательное?  
 push ax  
 mov al, ‘-’ ;да, надо вывести знак  
 int 29h  
 pop ax  
 neg ax ;преобразуем в положительное

L1:  
 xor cx, cx ;в cx будет кол-во знаков в строке  
 mov bx, 10 ;предпологаем десятичное число

Get\_next\_digit:

Xor dx, dx ;очищаем от остатка  
 div bx  
 push dx  
 inc cx ;еще один знак  
 or ax, ax  
 jnz get\_next\_digit

Print\_digit:  
 pop ax ;извлекаем остаток из стека

….(остальное я уснул и не переписал☹) (было ещё что-то со структурами)

**МАТЕМАТИЧЕСКИЙ СОПРОЦЕССОР 8087.**

Числа с плавающей запятой:  
1. Одинарная точность: 1 бит 8 бит 23 бита (Зн, порядок, мантисса)

2. Двойная точность: 1 бит 11 бит 52 бит

3. Расширенная точность: 1 бит 15 бит 64 бит

Положительный ноль: 0 0…0 0…0  
Отрицательный ноль: 1 0…0 0…0  
Наименьшее положительное число: 0 0…01 0…0  
Наименьшее отрицательное число: 1 0…01 0…0  
Наибольшее положительное(отрицательное): 0(1) 11..10 1..1  
Положительная бесконечность(отрицательное): 0(1) 1…1 0…0  
Нечисло: 1 1..1 x…x  
Неопределенность: 1 1..1 10..00

Блок FPU может выполнять одну операцию с плавающей точкой за один такт.  
Сопроцессор использует теги, чтобы определить переполнение стека или считывание с вершины стека, но стек пустой.

Структура сопроцессора 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Регистр | Знак | Экспонента | Мантисса |  |
| R7 |  |  |  |  |
| R6 |  |  |  |  |
| R5 |  |  |  | ST(2) |
| R4 |  |  |  | ST(1) |
| R3 |  |  |  | ST(0) |
| R2 |  |  |  |  |
| R1 |  |  |  |  |
| R0 |  |  |  |  |

|  |
| --- |
| Тег |
|  |
| 1\_\_\_\_\_\_0 |

|  |
| --- |
| Регистр управления |
| Регистр состояния TOS=011 |
| Слово тегов |
| 15\_\_\_\_\_\_\_\_\_\_\_\_\_\_0 |

|  |
| --- |
| Указатель команды |
| Указатель данных |

|  |
| --- |
| Регистр управления |
| Регистр состояния |

|  |
| --- |
| АЛУ для порядка числа |

**Операционный блок**

15 0

**Блок управления**

|  |
| --- |
| Сдвигатель |

A19-10

|  |
| --- |
| Управление интерфейсом |

|  |
| --- |
| Команда |
| Очередь команд |

|  |
| --- |
| АЛУ для мантиссы |

S2-0

Тег

AD15-0

|  |
| --- |
|  |
|  |
|  |
|  |

|  |
| --- |
| Указатель команды |
| Указатель данных |

|  |
| --- |
|  |
|  |
| Регистровый стек |
|  |

R0

R1

R7

79 0

1 0

31 0

**ОБРАБОТЧИКИ ПРЕРЫВАНИЙ.**

->Обработчики прерываний внешних устройств(аппаратные прерывания).