

1. Архитектура МП 8088 и 80386. Образование физического адреса.	4
2. Характеристики регистров	4
3. Флаги.	6
4. Сегментные регистры по умолчанию.	6
5. Структура одномодульной программы MS DOS. Повторные описания сегментов.	7
6. Возможные структуры кодового сегмента.	8
7. Возможные способы начала выполнения и завершения программы MS DOS.	8
8. Структура программы из нескольких исходных модулей MS DOS	9
9. Стандартные директивы описания сегментов.	10
10. Переменные, метки, символические имена и их атрибуты.	10
11. Виды предложений языка Ассемблер.	11
12. Директивы (псевдооператоры): назначение и формы записи.	11
13. Возможные комбинации сегментов и умолчания.	12
14. Директивы ASSUME, ORG, END.	12
15. Структура процедур.	13
16. Внешние имена.	14
17. Типы данных и задание начальных значений.	14
18. Способы описания меток, типы меток.	15
19. Команды условных переходов.	15
20. Команды организации циклов.	15
21. Способы адресации.	16
22. Организация рекурсивных подпрограмм.	17
23. Арифметические команды	18
24. Связывание подпрограмм.	18
25. Команды CALL и RET.	18
26. Способы передачи параметров подпрограмм.	19
27. Способы сохранения и восстановления состояния вызывающей программы.	19
28. Конвенции языков высокого уровня.	20
29. Команды сдвига.	20
30. Команды логических операций.	21
31. Команды обработки строк и префиксы повторения.	21
32. Команды пересылки строк.	22
33. Команды сравнения строк.	23
34. Команды сканирования строк.	24
35. Команды загрузки строк.	24
36. Команды сохранения строк.	25
37. Листинг программы.	25

38. Макросредства.	25
39. Макроопределения (макрофункций и макропроцедур) и макрокоманды.	26
40. Директива INCLUDE и LOCAL.	27
41. Рекурсия в макроопределениях.	27
42. Параметры в макросах.	28
43. Директивы условного ассемблирования и связанные с ними конструкции.	28
44. Директивы IFB и IFNB в макроопределениях.	28
45. Директивы IFIDN и IFDIF в макроопределениях.	29
46. Операции ;; % & <> ! в макроопределениях.	29
;; — комментарий.	29
47. Блоки повторения REPT, IRP/FOR, IRPC/FORC, WHILE.	29
48. Директива EQU и = в MASM.	30
49. Директива TEXTEQU в MASM32.	30
50. Типы макроданных text и number.	31
51. Именованные макроконстанты MASM32	31
52. Макроимена, числовые и текстовые макроконстанты.	31
53. Директивы echo и %echo	31
54. Способы вывода значений макропеременных и макроконстант с пояснениями	31
55. Операции в выражениях, вычисляемых препроцессором MASM:	32
56. Подготовка ассемблерных объектных модулей средствами командной строки для использования в средах разработки консольных приложений на ЯВУ.	32
57. Добавление ассемблерных модулей в проект консольного приложения C.	33
58. Добавление ассемблерных модулей в проект консольного приложения PASCAL.	33
59. Использование ассемблерных вставок в модулях на ЯВУ.	33
60. Вызов подпрограммы C из ассемблерной в VS C++.	33
61. Передача глобальных данных, определённых в консольной программе VS C++, в ассемблерный модуль.	33
62. Передача глобальных данных, определённых в ассемблерном модуле в консольный модуль C.	34
63. Средства отладки в CodeView. Примеры.	34
64. Средства отладки в средах разработки консольных приложений на ЯВУ.	35
65. Способы адресации. Термины и смысл.	36
66. Связывание подпрограмм. Конвенции C, PASCAL, STDCALL, РЕГИСТРОВАЯ.	37
67. Многострочные макроопределения. Формальные и фактические параметры.	38
68. Многострочные макроопределения и директивы условного ассемблирования	38
69. Многострочные макроопределения и директивы генерации ошибок	39
70. Многострочные макроопределения и операции в них.	40
71. Макропроцедуры. Определения и вызовы	40

72. Рекурсивные макропроцедуры	40
73. Макрофункции. Определения и вызовы	41
74. Рекурсивные макроопределения	41
75. Числовые макроконстанты. Определение и примеры использования	42
76. Числовые макропеременные. Определение и примеры использования	42
77. Константные выражения. Операции, вычисляемые препроцессором в выражениях	42
78. Текстовые макропеременные. Способы определения	42
79. Блоки повторения REPT, FOR, FORC, WHILE.	42
80. Стандартные макрофункции обработки строк @CATSTR и @SUBSTR.	43
81. Инструменты отладки макросредств. Окно командной строки, листинг, сообщения препроцессора	43
82. Применение макросредств при определении переменных	44
83. Применение макросредств при создании фрагментов кода	45

1. Архитектура МП 8088 и 80386. Образование физического адреса.

8086 — 16-разрядный микропроцессор. Имеет 14 16-разрядных регистров, 16-разрядную шину данных, 20-разрядную шину адреса (поддерживает адресацию до 1 Мб памяти), поддерживает 98 инструкций. 8088 отличается от 8086 только тем, что у него 8-разрядная шина данных.

80386 — расширение архитектуры 8086 до 32-разрядной. Появился защищенный режим, позволяющий 32-битную адресацию; все регистры, кроме сегментных, расширились до 32 бит (приставка Е, н-р ЕАХ); добавлены два дополнительных 16-разрядных сегментных регистра FS и GS, а также несколько специальных 32-битных регистров (CRx, TRx, DRx). В защищенном режиме используется другая модель памяти.

Память представляет собой линейную последовательность байтов, поделенную на параграфы (16 идущих подряд байт).

Параграф является минимальным возможным в x86 сегментом. Полный физический адрес составляется из номера сегмента и смещения относительно начала этого сегмента (“байт 5 сегмента 3”).

<полный адрес> = <номер сегмента> * <размер сегмента> + <смещение>

2. Характеристики регистров

Регистр процессора — сверхбыстрая память внутри процессора, предназначенная для хранения адресов и промежуточных результатов вычислений (регистр общего назначения/регистр данных) или данных, необходимых для работы самого процессора (указатель команды).

Всего архитектуры 8086 и 8088 содержат 14 16-битных регистров:

Регистры общего назначения	Аккумулятор	AX	
		AH	AL
	База	BX	
		BH	BL
	Счётчик	CX	
		CH	CL
	Данные	DX	
		DH	DL
Указательные регистры	Указатель базы	BP	
	Указатель стека	SP	
Указатель команды	Указатель команды	IP	
Индексные регистры	Индекс источника	SI	
	Индекс получателя	DI	
Регистр состояния	Флаги	<div> <div>1514131211109876543210</div> <div> <div>— — — — OF DF IF TF SF ZF — AF — PF — CF</div> </div> </div>	
Сегментные регистры	Сегмент кода	CS	
	Сегмент данных	DS	
	Дополнительный сегмент	ES	
	Сегмент стека	SS	

Регистры общего назначения	Аккумулятор	EAX
		AX
		AH AL
	База	EBX
		BX
		BH BL
	Счётчик	ECX
		CX
		CH CL
	Данные	EDX
		DX
		DH DL
Указательные регистры	Указатель базы	EBP
		BP
	Указатель стека	ESP
		SP
Указатель команды	Указатель команды	EIP
		IP
Индексные регистры	Индекс источника	ESI
		SI
	Индекс получателя	EDI
		DI
Регистр состояния	Флаги	EFLAGS 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 Резерв для INTEL VM R F 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 - NT IOPL OF DF IF TF SF ZF - AF - PF - CF
Сегментные регистры	Сегмент кода	CS (16 бит)
	Сегмент данных	DS (16 бит)
	Дополнительный сегмент	ES (16 бит)
	F сегмент	FS (16 бит)
	G сегмент	GS (16 бит)
	Сегмент стека	SS (16 бит)
Специальные регистры	Управляющий регистр машины	CR0
	Резервный регистр для INTEL	CR1
	Линейный адрес прерывания из-за отсутствия страницы	CR2
	Базовый регистр таблицы страниц	CR3
	Регистр сегмента состояния задачи	TSSR
	Регистр таблицы глоб. дескриптора	GDTR
	Регистр таблицы дескр. прерывания	IDTR
	Регистр таблицы лок. дескриптора	LDTR
	и т.д.	...

При этом, например, AL нижние 8 бит регистра AX, AH верхние (это не отдельные регистры!).

В архитектуре 80386 все регистры, кроме сегментных, были расширены до 32 бит (приставка E), при этом, например, AX стал нижней половиной EAX и т.д., а так же были добавлены два новых 16битных сегментных регистра FS и GS и несколько специальных регистров (CRx, DRx, TRx).

3. Флаги.

Всего 12 флагов (используемых битов) в регистре FLAGS, 2 в регистре EFLAGS. Старший бит 31-ый в случае двойного слова (отсчет с нуля), 15-ый в случае слова, 7й в случае байта. Под “результат” имеется в виду результат последней операции/команды. Регистр FLAGS кладется в стек и берется из него командами PUSHF и POPF, EFLAGS командами PUSHFD, POPFD. Флаги, помеченные в таблице серым цветом, существуют начиная с архитектуры 80386.

Бит 0. CF (Carry Flag) - флаг переноса; = 1 при переносе из/заёме в (при вычитании) старший бит результата и показывает наличие переполнения в беззнаковой целочисленной арифметике.	Бит 10. DF (Direction Flag) - флаг направления; если = 0, то команды вроде MOVS будут увеличивать текущий адрес (идти по памяти слева направо), иначе - уменьшать (справа налево).
Бит 2. PF (Parity Flag) - флаг четности; = 1 если младший байт результата содержит чётное число единичных битов.	Бит 11. OF (Overflow Flag) - флаг переполнения; = 1, если целочисленный результат слишком длинный для размещения в целевом операнде (регистре или ячейке памяти), т.е. переполнение.
Бит 4. AF (Aux. Carry Flag) - доп. флаг переноса; = 1 при переносе или заёме из бита 4 результата.	Биты 12, 13. IOPL (I/O Privilege Level) - уровень приоритета ввода-вывода;
Бит 6. ZF (Zero Flag) - флаг нуля; = 1 если результат последней операции = 0.	Бит 14. NT (Nested Task) - флаг вложенности задач; = 1, если текущая задача вложена в другую.
Бит 7. SF (Sign Flag) - флаг знака; = значению старшего бита результата (в знаковой арифметике старший бит - знаковый: если = 1, то число < 0)	Бит 16. RF (Resume Flag) - флаг возобновления; флаг маскирования ошибок отладки.
Бит 8. TF (Trap Flag) - флаг трассировки; если = 1, разрешена пошаговая отладка.	Бит 17. VM (Virtual 8086 Mode Flag) - установка в защищенном режиме вызывает переход в режим виртуального 8086.
Бит 9. IF (Interrupt Enable Flag) - если = 1, прерывания разрешены (устанавливается и снимается командами STI и CLI)	Биты 18-31. Резерв для расширений Intel.

4. Сегментные регистры по умолчанию.

Сегментные регистры (CS, DS, ES, SS) нужны для хранения номера сегмента. Они используются вместе с адресными регистрами (SI, DI, IP, SP), которые указывают на смещение относительно начала сегмента, для получения полного адреса.

CS (регистр сегмента кода) используется с IP для адресации инструкций программы (IP указывает на следующую к выполнению команду), при этом по умолчанию при выполнении команд передачи управления (н-р J MP) их операнд воспринимается как смещение относительно текущего сегмента (ближний адрес). Можно передавать управление и в другой сегмент (дальний адрес: номер сегмента и смещение).

DS (регистр сегмента данных) используется по умолчанию для адресации данных (н-р MOV A, AX преобразуется в MOV DS:A, AX).

ES (регистр доп. сегмента) используется для тех же целей, если требуется доп. сегмент данных или связь приемник-передатчик (DS:SI → ES:DI, например).

Требуется некоторыми командами (н-р MOVS) .

SS (регистр сегмента стека) вместе с SP (указателем стека) используется для организации стека. SP при запуске программы содержит размер стека, если он не равен 0 или 64, и 0 в противном случае.

SS и CS устанавливаются автоматически при запуске программы, DS и ES надо устанавливать вручную.

5. Структура одномодульной программы MS DOS. Повторные описания сегментов.

```
SEG_STACK SEGMENT PARA STACK 'STACK'  
    DB 64 DUP('STACK')  
SEG_STACK ENDS
```

```
SEG_DATA SEGMENT PARA 'DATA'  
    S DW 42  
SEG_DATA ENDS
```

```
SEG_CODE SEGMENT PARA 'CODE'  
    ASSUME CS:SEG_CODE, DS:SEG_DATA, SS:SEG_STACK
```

```
PROGSTART:  
    MOV AX, SEG_DATA  
    MOV DS, AX  
    ; ... DO Smth  
    MOV AH, 4Ch  
    INT 21h  
SEG_CODE ENDS  
END PROGSTART
```

Повторные описания сегментов обычно используются, когда один и тот же сегмент описывается в нескольких модулях, или чтобы расположить данные рядом с операциями над ними. При этом в случае разных модулей указываются видимые с наружи(в других модулях) элементы (PUBLIC name) , а в модулях , где эти элементы надо увидеть, в том же сегменте указываются их имена и тип (EXTRN name: type)

```
; МОДУЛЬ 1  
SEG_DATA SEGMENT PARA 'DATA'  
PUBLIC A  
EXTRN S: BYTE  
    A DW 666  
SEG_DATA ENDS
```

```
; МОДУЛЬ 2  
SEG_DATA SEGMENT PARA 'DATA'  
PUBLIC S  
EXTRN A: WORD  
    S DB 42  
SEG_DATA ENDS
```

6. Возможные структуры кодового сегмента.

```
ИМЯ_СЕГМЕНТА_КОДА SEGMENT [<вид_выравнивания>] ['CODE']  
    ASSUME CS:ИМЯ_СЕГМЕНТА_КОДА, DS:ИМЯ_СЕГМЕНТА_ДАННЫХ, SS:  
ИМЯ_СЕГМЕНТА_СТЕКА
```

```
ПРОЦ1 PROC [FAR|NEAR]
```

```
...
```

```
    CALL ПРОЦ2
```

```
...
```

```
    CALL ПРОЦ3
```

```
...
```

```
    RET|RETF
```

```
ПРОЦ1 ENDP
```

```
ПРОЦ2 PROC
```

```
...
```

```
    RET
```

```
ПРОЦ2 ENDP
```

```
ПРОЦ3 PROC FAR
```

```
...
```

```
    RET ; ЭТОТ RET АВТОМАТИЧЕСКИ ЗАМЕНИТСЯ НА RETF  
ПРОЦ3 ENDP
```

```
...
```

```
МЕТКА_ТОЧКИ_ВХОДА:
```

```
    ; ...
```

```
ИМЯ_СЕГМЕНТА_КОДА ENDS
```

```
END МЕТКА_ТОЧКИ_ВХОДА
```

FAR — подпрограмма дальнего вызова (можно вызывать из других сегментов),
NEAR — ближнего (вызов только из этого сегмента).

RET — возврат из подпрограммы ближнего, RETF — дальнего вызова. В отличие от обычного RET'а, который вытаскивает из стека только значение IP, RETF вытаскивает из стека значения CS и IP.

По умолчанию стоит NEAR.

<вид_выравнивания> может быть PARA, BYTE, WORD, PAGE. Указывает, чему должны быть кратны адрес начала и конца сегмента. По умолчанию PARA.

7. Возможные способы начала выполнения и завершения программы MS DOS.

Способы входа: END <метка в кодовом сегменте>, END <процедура(far)>

Способы выхода: при помощи MOV AH,4Ch // INT 21h и при помощи RET'а, если программа началась со входа в подпрограмму.

При этом если выход с помощью RET, то на верхушке стека должен лежать 0, а за ним DS, положенный туда в начале программы, а AX должен быть 0. Т.к. при старте программы DOS кладет в DS адрес своей системной структуры, называемой Program Segment Prefix (PSP). Эта структура содержит аргументы командной строки, адрес возврата в DOS и прочую информацию. Первые два байта ее ВСЕГДА равны машинному коду команды INT 20h. Поэтому, когда мы кладем в стек DS, а затем 0 прямо в самом начале программы, по RET(F) в CS положится адрес PSP, а в IP 0. А нулевая инструкция в этом нашем новом CS и есть INT 20h, которая является одним из прерываний, завершающих программу в DOS, и она будет выполнена процессором следующей после RET. Это прерывание берет из AX код ошибки (0 = нет ошибки).

8. Структура программы из нескольких исходных модулей MS DOS

Сначала создаются объектики отдельных модулей с помощью компилятора (masm.exe), а затем всё компоуется в экзешник линковщиком (link.exe).

```
; МОДУЛЬ 1 ГЛАВНЫЙ  
PUBLIC SHIT1
```

```
DSEG SEGMENT PARA PUBLIC 'DATA'  
    ASS DB 22  
DSEG ENDS
```

```
SSEG SEGMENT PARA STACK 'STACK'  
    DB 64 DUP ('ASS-')  
SSEG ENDS
```

```
CSEG SEGMENT PARA PUBLIC 'CODE'  
    ASSUME CS:CSEG, DS:DSEG, SS:SSEG  
    EXTRN SHIT2:NEAR
```

```
SHIT1 PROC FAR  
    CALL SHIT2  
    MOV AH, 4Ch  
    INT 21h  
SHIT1 ENDP
```

```
CSEG ENDS  
END SHIT1 ;Это определяет точку входа во всю программу
```

```
; МОДУЛЬ 2  
DSEG2 SEGMENT PARA PUBLIC 'DATA'  
    FUCK DB "HELLO AVGN!$"   
DSEG2 ENDS  
EXTRN ASS:BYTE
```

```
CSEG SEGMENT PARA PUBLIC 'CODE'
    ASSUME CS:CSEG, DS:DSEG2
    PUBLIC SHIT2
```

```
    SHIT2 PROC NEAR
        MOV AX, SEG ASS
        MOV ES, AX
        MOV AH, 09h
        MOV DX, FUCK
        INT 21h
        INC ES:ASS
        RET
    SHIT2 ENDP
CSEG ENDS
END ; модуль неглавный
```

9. Стандартные директивы описания сегментов.

```
<ИМЯ_СЕГМЕНТА> SEGMENT [<ВЫРАВНИВАНИЕ>] [<ТИП>] [<КЛАСС>] ...
<ИМЯ_СЕГМЕНТА> ENDS
```

<ВЫРАВНИВАНИЕ>:

- BYTE без
- WORD по границе слова
- PARA по границе параграфа
- PAGE по 256 байт (странице)

<ТИП>:

- PUBLIC объединение сегментов с одним именем
- STACK сегмент является частью стека
- COMMON расположение на одном адресе с другими COMMON-сегментами
- AT расположить по абсолютному адресу ADDR

<КЛАСС>:

- идентификатор ('STACK', 'CODE', 'DATA', ...)

10. Переменные, метки, символические имена и их атрибуты.

Конструкции ассемблера формируются из идентификаторов и ограничителей.

Идентификатор — набор символов (буквы, цифры, “_”, “.”, “?”, “\$”, “@”).

Символ “.” может находиться только в начале идентификатора. Идентификатор не может начинаться с цифры, может размещаться только на одной строке и содержит только от 1 до 31 символа. С помощью идентификаторов можно представить переменные, метки и имена.

Переменные идентифицируют хранящиеся в памяти данные и имеют три атрибута:

- Сегмент
- Смещение
- Тип (DB(1 байт), DW(2), DD(4), DF/DP(6), DQ(8), DT(10))

Метка — частный случай переменной. На неё можно ссылаться посредством переходов и вызовов.

Имеет два атрибута:

- Сегмент
- Смещение

Символические имена — символы, определённые директивой EQU, имеющие значение типа “символ” или “число”.

DB ? ; Просто выделена память в сегменте под 1 байт
 DB 64 DUB(B,C,D) ; 64 байта, заполненные поочерёдно B,C и D
 A DB 123 ; Переменная A
 B EQU 456 ; Константа B
 C = 902 ; Константа C, которую можно переобъявлять

11. Виды предложений языка Ассемблер.

Предложения бывают трех типов:

1. Команды (и их операнды) — символические аналоги машинных инструкций (мнемоники). В процессе трансляции они преобразуются в машинный код ассемблером (н-р MOV AX,0)
2. Директивы — указания ассемблеру на выполнение определенных действий; не имеют аналога в машинном коде (н-р A EQU 9) .
3. Комментарии — строка или часть строки из любых символов, начинающаяся с символа ‘;’.

12. Директивы (псевдооператоры): назначение и формы записи.

Директивы:

Директивы — указания ассемблеру на выполнение определенных действий; не имеют аналога в машинном коде.

- Директивы определения набора инструкций: н-р, строка **.386** в начале файла с кодом говорит ассемблеру использовать только наборы команд из архитектуры 80386 и ниже
- Упрощенные директивы сегментации
 - .MODEL <модельадр> [<модиф.адр.>] [,<язык>] [,<модиф.языка>]
 :
 - ♣ <модельадр> : модель адресации (FLAT 32битная стандартная)
 - ♣ <модиф.адр>:тип адресации: use16|use32|dos
 - ♣ <язык>, <модиф.языка>: определяют особенности соглашений о вызовах

- **.CODE[<имяseg>]**: задает положение начала сегмента кода
- **.STACK[<размер>]**: задает размер стека. По умолчанию — 10246
- **.DATA**: начало или продолжение сегмента инициализированных данных
- **.DATA?**: начало или продолжение сегмента неиниц. данных
- **.CONST**: начало или продолжение сегмента констант

Также директивы объявления данных DB, DW, DD, ...

Псевдооператоры:

<ИМЯ_ИДЕНТИФИКАТОРА> EQU <ВЫРАЖЕНИЕ>: используется для задания констант. Например, **ASS EQU 10** вызовет последующую замену каждого слова ASS в коде на 10 транслятором. Замена “тупая”, то есть во всем тексте программы каждое вхождение того что слева будет заменено на то что справа. Выражения могут быть и текстовые, и числовые, и даже куски текста программы. **<ИМЯИДЕНТИФИКАТОРА> = <ЧИСЛОВОВЫРАЖЕНИЕ>**: то же, что и **EQU**, но константы, заданные =, можно переопределять позднее в коде и работает он только для числовых выражений. При этом числовое выражение может включать что угодно, что может в числовом виде вычислить компилятор, н-р: 3+2 , ASS+4 (если ASS — заданная выше числовая константа).

13. Возможные комбинации сегментов и умолчания.

Если комбинация не указана, то по умолчанию сегменты считаются *разными*, **даже если у них одно имя и класс**. Комбинировать объявления сегментов (сегментные директивы) можно с помощью указания ТИПА сегмента.

14. Директивы ASSUME, ORG, END.

ASSUME

ASSUME <СЕГ_РЕГ1>:<ИМЯ_СЕГ1>[, <СЕГ_РЕГ2>:<ИМЯ_СЕГ2>[, . . .]]

Пример: **ASSUME DS:DATA_SEG, ES:EXTRA_SEG** Директива специализации сегментов. Обычно пишется в сегменте кода второй строчкой. Пример сообщает ассемблеру, что для сегментирования адресов из сегмента DATA_SEG выбирается регистр DS, из EXTRA_SEG — регистр ES. Теперь префиксы DS: , ES: при использовании переменных из этих сегментов можно опускать, и ассемблер их будет ставить самостоятельно.

ORG

Директива org позволяет задать компилятору начальный адрес в пределах сегментов кода, данных и EEPROM-памяти. В случае применения в сегменте кода, директива определяет адрес размещения 16-разрядного слова программ.

Синтаксис написания:

org {начальный адрес}

ORG 100h

END

CSEG SEGMENT PARA PUBLIC 'CODE'

; ...

```
START:
; ...
CSEG ENDS
END START
```

Здесь директива END указывает на то, что точка входа в программу находится на метке START, а так же на окончание файла с исходным кодом. Строки после нее игнорируются. Если меток в коде нет, можно после END ничего не указывать. Если опустить END, MASM выдаст ошибку.

15. Структура процедур.

Вызов процедуры производится при помощи инструкции CALL. Если процедура NEAR, то в стек заносится только значение IP следующей за CALL'ом инструкции. Если процедура FAR, то дополнительно в стек заносится CS, так как FAR процедура обычно лежит в другом кодовом сегменте.

```
...
CALL A ; После выполнения этой строчки в стеке будет лежать адрес CS:[MOV AX,
BX]
MOV AX,BX
...
```

```
; Общий вид процедуры
ИМЯ_ПРОЦ PROC [NEAR|FAR]
```

```
...
RET ; Если PROC FAR, то ассемблер заменит RET на RETF
ИМЯ_ПРОЦ ENDP
```

Возможное начало и завершение процедуры, которой передается управление при старте программы

- Возврат командой RET
 - Процедура должна быть дальней

ИМЯ_ПРОЦ PROC FAR ;дальняя процедура

```
PUSH DS ; помещаем в стек
MOV AX,0 ;сегментную часть PSP
PUSH AX ; и число 0
```

```
MOV AX, ИмяСегментаДанных ; настройка DS на
MOV DS,AX ;сегмент данных
```

ASSUME DS: ИмяСегментаДанных ;помогаем ассемблеру разобраться с сегментами

```
...
RET ;тоже дальняя команда
ИМЯ_ПРОЦ ENDP
```

16. Внешние имена.

Объявление внешних имён производится при помощи директивы PUBLIC <имя>, а декларация производится при помощи директивы EXTRN <имя>:<тип>. Работает на метках, именах процедур и переменных/константах.

; Модуль 1

...

A DW21

PUBLIC A ;Теперь A доступна из других модулей

; Модуль 2

...

EXTRN A:WORD ;Объявление внешней переменной, находящейся в другом файле

...

17. Типы данных и задание начальных значений.

- Целые числа:
 - XXX обычно десятичное
 - XXXb(BIN) - в двоичной системе
 - XXXq(OCT) - в восьмеричной системе(q, чтобы не спутать с 0)
 - XXXo(OCT) - в восьмеричной системе
 - XXXd(DEC) - в десятичной системе
 - XXXh(HEX) - в шестнадцатеричной системе
- Символы и строки (символ — один char, строка — 2 и более):
 - 'Символы'
 - "Символы"(Строки в DOS'е \$-терминированные)
- Структуры

Начальное значение может быть неопределённым. В таком случае вместо начального значения ставится "?".

a DB 12

a DB 0Ch

a DB 00001100b

a DB 14q

a DD 20 DUP (0)

описывает массив a из 20 элементов, начальные значения которых равны 0.

Символьная строка, предназначенная для корректного вывода, должна заканчиваться нуль-символом '\0' с кодом, равным 0.

Str DB 'Привет всем!', 0

Для перевода строки могут использоваться символы

- возврат каретки с кодом 13 (0Dh)
- перевод строки с кодом 10 (0Ah).

Stroka DB «Привет», 13, 10, 0

18. Способы описания меток, типы меток.

Метка - это символьное имя, обозначающее ячейку памяти, которая содержит некоторую команду.

Метку в программе можно задать двумя способами:

1. <имя>:<команда> (только ближние метки)
2. <имя>:LABEL <тип метки> (любую метку)

Метка имеет **атрибуты: сегмент, смещение, тип:**

NEAR – ближняя метка

FAR – дальняя метка. Переход возможен также из другого сегмента.

19. Команды условных переходов.

Имеется 19 команд, имеющих 30 мнемонических кодов операций. Команды проверяют состояние отдельных флагов или их комбинаций, и при выполнении условия передают управление по адресу, находящемуся в операнде команды, иначе следующей команде.

Команда	Мнемоника	С чем работает	Прыжок, когда
JE JZ	Equal (Zero)	все	ZF == 1
JNE JNZ	Not Equal (Not Zero)	все	ZF == 0
JG JNLE	Greather	ЦЗЗ	(ZF == 0) && (SF == OF)
JGE JNL	Greather or Equal	ЦЗЗ	SF == OF
JL JNGE	Less	ЦЗЗ	SF != OF
JLE JNG	Less or Equal	ЦЗЗ	(ZF == 1) (SF != OF)
JA JNBE	Above	ЦБЗ	(CF == 0) && (ZF == 0)
JAE JNB	Above or Equal	ЦБЗ	CF == 0
JB JNAE	Below	ЦБЗ	CF == 1
JBE JNA	Below or Equal	ЦБЗ	(CF == 1) (ZF == 1)
JO	Overflow	все	OF == 1
JNO	Not overflow	все	OF == 0
JC	Carry	все	CF == 1
JNC	Not Carry	все	CF == 0
JS	Sign (< 0)	ЦЗЗ	SF == 1
JNS	Not Sign (>= 0)	ЦЗЗ	SF == 0
JP	Parity	все	PF == 1
JNP	Not Parity	все	PF == 0
JCXZ	CX is Zero	все	CX == 0

20. Команды организации циклов.

LOOP x метка(параметр), где **метка(параметр)** — определяет адрес перехода от 128 до +127 от команды LOOP.

Мнемоника	Описание
LOOP	--CX. Если (CX != 0) — повторяем цикл (переходим к метке, указанной в команде)
LOOPE LOOPZ	--CX. Если (CX != 0) && (ZF == 1) — повторяем цикл.
LOOPNE LOOPNZ	--CX. Если (CX != 0) && (ZF == 0) — повторяем цикл.

Примеры использования:

Обычный цикл:

Ассемблерный код	Аналог на С (просто для понимания)
<pre>MOV CX, 5 SUMMATOR: ADD AX, 10 LOOP SUMMATOR</pre>	<pre>for (int CX = 5; CX > 0; --CX) AX += 10;</pre>

Вложенный цикл:

Ассемблерный код	Аналог на С (просто для понимания)
<pre>MOV CX, 5 SUMAX: PUSH CX ; Сохранение в стеке CX ; внешнего цикла ADD AX, 10 MOV CX, 7 SUMBX: ADD BX, 10 LOOP SUMBX POP CX LOOP SUMAX</pre>	<pre>for (int CX = 5; CX > 0; --CX) { AX += 10; // Здесь вместо запушивания я юзаю // другую переменную for (int CX2 = 7; CX2 > 0; --CX2) BX += 10; }</pre>

21. Способы адресации.

В инструкциях программ операнды строятся из объектов, представляемых собой имена переменных и меток, представленных своим сегментом смещения (регистры, числа, символические имена с численным значением). Запись этих объектов с использованием символов: [], +, -, . в различных комбинациях называют способами адресации.

- 1. Непосредственная адресация (значение):** операнд задается непосредственно в инструкции и после трансляции входит в команду, как составная часть 1 и 2 байта.

`MOV AL, 2` ;(1байт)

`MOV BX, 0FFFFH` ;(2байта)

- 2. Регистровая адресация:** значение операнда находится в регистре, который указывается в качестве аргумента инструкции.

`MOV AX, BX`

- 3. Прямая адресация:** в инструкции используется имя переменной, значение которой является операндом.

`MOV AX, X` ; в AX значение X

`MOV AX, CS:Y` ; в AX CS по смещению Y

- 4. Косвенная регистровая адресация:** в регистре, указанном в инструкции, хранится смещение (в сегменте) переменной, значение которой и является операндом. Имя регистра записывается в [].

`MOV BX, OFFSET Z`

...

MOV BYTE PTR[BX], 'ASS' ; в байт, адресованный регистром BX, запомнить 'ASS' в сегменте DS

MOV AX, [BX] ; в AX запомнить слово из сегмента, адрес BX со смещением DS

MOV BX, CS:[SI] ; записать данные в BX, находящиеся в CS по смещению SI

- 5. Косвенная базово-индексная адресация (адресация с индексацией и базированием):** смещение ячейки памяти, где хранятся значения операндов, вычисляется, как сумма значений регистров, записанных в качестве параметра в инструкции. $[R1+R2]$, где $R1 \in \{BX, BP\}$, $R2 \in \{SI, DI\}$

MOV AX, [BX + DI]

MOV AX, [BP + SI]

Для BX по умолчанию сегменты из регистра DS. Для BP по умолчанию сегменты из регистра SS.

- 6. [Косвенная] базово-индексная адресация со смещением :** (прямая с базированием и индексированием), отличается от п.5 тем, что ещё прибавляется смещение, которое может быть представлено переменной или ЦБЗ.

; $[R1 + R2 \pm \text{ЦБЗ}]$

; $[R1][R2 \pm \text{ЦБЗ}]$

; имя $[R1 + R2 \pm \text{Абс. выражение}]$

; имя $[R1][R2 \pm \text{Абс. выражение}]$

MOV AX, ARR[EBX][ECX * 2]

; $\text{ARR} + (\text{EBX}) + (\text{ECX}) * 2$

; вот такой монстр

; $[\text{Имя}][\text{База}][\text{Индекс} [* \text{масштаб}]] [\text{Абс. выражение}]$

; Смещение операнда = Смещение имени + значение базы + значение индекса * масштаб + значение абсолютного выражения

22. Организация рекурсивных подпрограмм.

В программировании рекурсивной называется процедура, которая прямо или косвенно вызывает саму себя.

; Рекурсивная процедура вычисления факториала

; вход: CX – число без знака

; выход: AX – результат

factorial:

push bp

; Сохранение BP

mov bp, sp

; BP=SP

mov ax, [bp+4]

; AX=параметр

test ax, ax

; Проверка AX

jz f_ret1

; Если 0, вернуть 1

dec ax

; Декремент AX

push ax

; Помещение параметра в стек

call factorial

; Вызов процедуры для предыдущего числа

mul word[bp+4]

; Умножение результата на параметр процедуры

jmp f_ret

; Переход к возврату из процедуры

f_ret1:

inc ax

f_ret:

pop bp

; Восстановление BP

ret 2

; Возврат из процедуры

Основной недостаток рекурсии — это большое использование памяти в стеке. Рекурсия здорово ест ресурсы! При каждом вызове процедуры в стеке сохраняется адрес возврата, используемые регистры, параметры процедуры, а также может выделяться место для локальных переменных. Кроме того, на работу со стеком тратится дополнительное время, что может сделать рекурсию медленной.

23. Арифметические команды

NEG	Обратить знаковый бит (* -1)
ADD	Сложение операндов
SUB	Вычитание операндов
ADC	Сложение операндов + флаг переноса (<При> + <Ист> + <CF>)
SBB	Вычитание операндов - флаг переноса (<При> - <Ист> - <CF>)
MUL	Умножение (E)AX на операнд (для беззнаковых)
DIV	Деление (E)AX с остатком на операнд (для беззнаковых). В (E)AX кладётся - результат деления, в (E)DX - остаток
IMUL	MUL для знаковых
IDIV	DIV для знаковых

24. Связывание подпрограмм.

Связывание подпрограмм — совокупность действий по:

- передаче управления подпрограмме
- передаче параметров через регистры, общую область памяти, области параметров
- возврату управления из подпрограмм
- запоминанию и восстановлению состояния вызывающей программы

Все это — связывание подпрограмм в исполняемом модуле. В более широком плане, когда речь идет о динамическом вызове подпрограмм или программ, хранящиеся на диске в виде отдельных модулей, к этому перечню добавляется:

1. загрузка модуля подпрограмм в ОЗУ и разрешение внешних ссылок (по передаче управления и по передаче данных).
2. освобождение памяти, после завершения работы вызываемого модуля и возврат управления из вызываемого модуля.

25. Команды CALL и RET.

CALL передает управление в другую строку программы (выполняется вызов процедуры), при этом сохраняя в стеке адрес возврата IP для команды **RET** (он указывает на следующую после **CALL** инструкцию). Если переход в другой сегмент, также сохраняется текущий сегмент кода (CS), в который нужно будет вернуться.

Возврат из процедуры выполняется командой **RET**. Эта команда восстанавливает значение из вершины стека в регистр IP. Таким образом, выполнение программы продолжается с команды, следующей сразу после команды **CALL**. Обычно код процедуры заканчивается этой командой.

26. Способы передачи параметров подпрограмм.

Через регистры:

плюсы: быстро, просто

минусы: мало регистров, нужно постоянно следить за значениями в регистрах, регистры маленькие

Через общую область памяти, которая организуется при помощи COMMON сегментов:

плюсы: удобно возвращать большое кол-во данных

минусы: можно испортить данные извне + данные перекрываются в памяти, что не есть хорошо

Через общий стек (используется чаще всего):

плюсы: можно передавать большое количество параметров

минусы: количество параметров ограничено размерами стека и разрядностью ВР + требуется наличие стека

Через области параметров, адреса которых передаются через регистры: (по сути то же, что и 1, но передаем адреса начала списков параметров и может быть их количество)

плюсы: можно передавать большое количество параметров или даже переменное число параметров

минусы: должна быть заранее подготовленная под параметры и заполненная область памяти, надо иметь в виду, что вызываемый код может работать с другой областью памяти

27. Способы сохранения и восстановления состояния вызывающей программы.

Вариант 0:

Хранилище — область (стек) вызываемой программы.

Сохранитель (восстановитель) — вызывающая программа

“+” минимум затраты времени и памяти на сохранение, т.к. вызываемая программа знает, какие нужно сохранить регистры

“-” если много вызовов подпрограммы, то команды, выполняющие сохранение регистров, будут занимать значительную часть кода

Вариант 1:

Хранилище — область (стек) вызываемой программы.

Сохранитель (восстановитель) — вызываемая подпрограмма

“+” сокращение кода основной подпрограммы

“-” подпрограмма не знает, какие регистры сохранять, сохранение регистров описывается в специальном соглашении.

28. Конвенции языков высокого уровня.

Конвенция Pascal заключается в том, что параметры из программы на языке высокого уровня передаются в стеке и возвращаются в регистре AX/EAX, поместить параметры в стек в естественном порядке. За уничтожение параметров в стеке отвечает вызываемая программа.

Конвенция C. Параметры помещаются в стек в обратном порядке, и, в противоположность PASCAL-конвенции, удаление параметров из стека выполняет вызывающая процедура.

Тип возвращаемого значения	Регистр
<code>unsigned char</code>	al
<code>char</code>	al
<code>unsigned short</code>	ax
<code>short</code>	ax
<code>unsigned int</code>	eax
<code>int</code>	eax
<code>unsigned long int</code>	edx:eax
<code>long int</code>	edx:eax

Существует конвенция передачи параметров **STDCALL**, отличающаяся и от C, и от PASCAL-конвенций, которая применяется для всех системных функций Win32 API. Здесь параметры помещаются в стек в обратном порядке, как в C, но процедуры должны очищать стек сами, как в PASCAL.

Регистровое соглашение FASTCALL. В этом случае параметры в функции также передаются по возможности через регистры. Например, при вызове функции с шестью параметрами `some_proc(a,b,c,d,e,f);` первые три параметра передаются соответственно в EAX, EDX, ECX, а только начиная с четвертого, параметры помещают в стек в обычном обратном порядке. В случае если стек был задействован, освобождение его возлагается на вызываемую процедуру.

29. Команды сдвига.

команда операнд, количество_сдвигов

Мнемоника	Описание	Мнемоника	Описание
SHL/SHR	Логический сдвиг	ROR/ROL	Циклический сдвиг
SAL/SAR	Арифметический сдвиг	RCL/RCR	ROR/ROL с установкой CF

SHL и SHR сдвигают биты операнда (регистр/память) влево или вправо соответственно на один разряд и изменяют флаг переноса cf. При логическом сдвиге все биты равноправны, а освободившиеся биты заполняются нулями

Команда SAR — сдвигает биты операнда (регистр/память) вправо на один разряд, значение последнего вытолкнутого бита попадает в флаг переноса, а освободившиеся биты заполняются знаковым битом.

Команда SAL — сдвигает биты операнда (регистр/память) влево на один разряд, значение последнего вытолкнутого бита попадает в флаг переноса, а освободившиеся биты заполняются нулями, при этом знаковый бит не двигается.

Циклический сдвиг напоминает смещение, выдвигаемые биты, снова вдвигаются с другой стороны.

ROL и ROR сдвигают все биты операнда влево(для ROL) или вправо(для ROR) на один разряд, при этом старший(для ROL) или младший(для ROR) бит операнда вдвигается в операнд справа(для ROL) или слева(для ROR) и становится значением младшего(для ROL) или старшего(для ROR) бита операнда; одновременно выдвигаемый бит становится значением флага переноса cf.

RCL и RCR сдвигают все биты операнда влево (для RCL) или вправо (для RCR) на один разряд, при этом старший(для RCL) или младший(для RCR) бит становится значением флага переноса cf; одновременно старое значение флага переноса cf вдвигается в операнд справа(для RCL) или слева(для RCR) и становится значением младшего(для RCL) или старшего(для RCR) бита операнда.

30. Команды логических операций.

Команда	Описание
AND	Выполняет операцию логического И между двумя операндами
OR	Выполняет операцию логического ИЛИ между двумя операндами
XOR	Выполняет операцию исключающего ИЛИ между двумя операндами
NOT	Выполняет операцию логического отрицание (НЕ) единственного операнда
TEST	Выполняет операцию логического И между двумя операндами, устанавливает соответствующие флаги состояния процессора, но результат операции не записывается вместо операнда получателя данных

31. Команды обработки строк и префиксы повторения.

Этим командами могут обрабатываться строки (последовательности байтов или слов). Максимальная длина такой строки - 64 кб (так как регистры 16-битные). С ними часто используются префиксы повторения.

Существуют пять основных операций или примитивов, выполняющих обработку одного элемента - слова (байта) за один приём.

Команды-примитивы (текущий элемент = на него сейчас указывают SI или DI):

- - сравнение (CMPS): сравнивает поэлементно источник и приемник;
- - сканирование (SCAS): ищет значение AX(AL) в приемнике (сравнивает текущий элемент с AX(AL));
- - пересылка (MOVS): копирует поэлементно из источника в приемник

- - загрузка (LODS): копирует текущий элемент из источника в AX(AL);
- - сохранение в ОЗУ (STOS): заменяет текущий элемент в приемнике на AX(AL).

Каждая команда имеет 4 разновидности (на примере MOVS):

- - без суффикса размера элементов, но с явным указанием приёмника и источника: MOVS <ПРИ>, <ИСТ>
- - ключевое слово с суффиксом B и без параметров (команда побайтовой обработки): MOVSB
- - ключевое слово с суффиксом W и без параметров (команда обработки слов): MOVSW
- - ключевое слово с суффиксом D и без параметров (команда обработки двойных слов): MOVSD

В версиях этих команд без параметров считается, что источник находится в сегменте DS по смещению SI (DS:SI), приемник — в ES:DI. После выполнения команды автоматически изменяется значение в SI и DI, причём SI и DI возрастают на 1 (если в конце B), 2 (если в конце W) или 4 (если в конце D) при флаге DF = 0, уменьшаются на 1/2/4 при DF = 1.

Состоянием DF можно управлять командами CLD (DF=0) и STD (DF=1). В версии с параметрами 1 или 2 выбирается автоматически в зависимости от разрядности аргумента.

Префиксы повторения:

Описание	Мнемоника	Обычно используется с	Условие окончания
Повторять до CX=0	REP	MOVS, LODS, STOS	CX=0
Повторять, пока равно и CX≠0	REPE, REPZ	CMPS, SCAS	CX=0 или ZF=0
Повторять, пока не равно и CX≠0	REPNE, REPNZ	CMPS, SCAS	CX=0 или ZF=1

32. Команды пересылки строк.

Команды пересылки байтов MOVSB Функция:

а) при DF = 0 (установить командой CLD)

байт [DS:SI] → байт [ES:DI]

SI = SI + 1

DI = DI + 1

б) при DF = 1 (установить командой STD)

байт [DS:SI] → байт [ES:DI]

SI = SI - 1

DI = DI - 1

Флаги не меняются.

Пересылка слов MOVSW

Всё то же, но копируется по два байта (слово) и

а) SI = SI + 2

DI = DI + 2

б) $SI = SI - 2$

$DI = DI - 2$

Аналогично с двойными словами (4 байта)

Общая команда

`MOVS <Приемник>, <Источник>`

Функция: то же, что и команды выше, но для адресов начала источника и приемника, указанных в операндах:

а) то же, что и у `MOVSB`, если тип источника и приемника `BYTE`

б) то же, что и у `MOVSW`, если тип источника и приемника `WORD`,

в) то же, что и у `MOVSD`, если тип источника и приемника `DWORD`

Пример:

`P1 DD STR1`

`P2 DD STR2`

...

`LDS SI,P1 LES DI,P2 ...`

`MOV CX,5 REP MOVSB`

33. Команды сравнения строк.

Команды сравнения строк `CMPSB`, `CMPSW`, `CMPS`, `CMPSD`.

с суффиксом `B` - для обработки байтов,

с суффиксом `W` - для обработки слов,

с суффиксом `D` - для обработки двойных слов,

без суффикса - обработка байтов или слов определяется типом операндов.

2.1. Сравнение байт

Функция `CMPSB`:

- а) При $DF=0$ (исп. `CLD`)
 - Установка флагов `CF`, `PF`, `AF`, `ZF`, `SF`, `OF` по значению разности:
Байт[`DS:SI`]-Байт[`ES:DI`] (Источник - Приемник)
 - $SI:=SI+1$
 - $DI:=DI+1$
- б) При $DF=1$ (исп. `STD`)
 - Установка флагов `CF`, `PF`, `AF`, `ZF`, `SF`, `OF` по значению разности:
Байт[`DS:SI`]-Байт[`ES:DI`] (Источник - Приемник)
 - $SI:=SI-1$
 - $DI:=DI-1$

2.2. Сравнение слов

Функция `CMPSW`:

- а) При $DF=0$ (исп. `CLD`)
 - Установка флагов `CF`, `PF`, `AF`, `ZF`, `SF`, `OF` по значению разности:
Слово[`DS:SI`]-Слово[`ES:DI`] (Источник - Приемник)
 - $SI:=SI+2$
 - $DI:=DI+2$
- б) При $DF=1$ (исп. `STD`)

- Установка флагов CF, PF, AF, ZF, SF, OF по значению разности:
Слово[DS:SI]-Слово[ES:DI] (Источник - Приемник)
- SI:=SI-2
- DI:=DI-2

2.3. Сравнение байтов или слов

Функция CMPS Приемн,Источн:

- Та же, что у CMPSB, если тип Источн и Приемн = BYTE
- Та же, что у CMPSW, если тип Источн и Приемн = WORD

! В командах CMPS[B|W] флаги устанавливаются по разности Источн-Приемн, а не по Приемн-Источн, в отличие от CMP.

34. Команды сканирования строк.

Сканирование байтов **SCASB**

Функция: поиск байта AL в байтовой строке

а) DF = 0,
CMP AL, [ES:DI]
DI = DI + 1

б) DF = 1,
CMP AL, [ES:DI]
DI = DI - 1

Сканирование слов **SCASW**

Функция: то же самое, но сравниваются слова и с AX, а еще

- DI = DI + 2
- DI = DI - 2

Аналогично сканирование двойных слов и с EAX **SCASD**

Общая команда

SCAS <Приемник>

Функция: то же, что и команды выше, но в качестве начала приемника используется операнд:

- ~**SCASB**, если <Приемник> - BYTE
- ~**SCASW**, если <Приемник> - WORD
- ~**SCASD**, если <Приемник> - DWORD

35. Команды загрузки строк.

Загрузка байтов **LODSB**

Функция: загрузка текущего байта источника в AL

а) $DF = 0$

байт $[DS:SI] \rightarrow AL$

$SI = SI + 1$

б) $DF = 1$

байт $[DS:SI] \rightarrow AL$

$SI = SI - 1$

Загрузка слов **LODSW**

Функция: то же, что и выше, но с **AX** и словами и

а) $SI = SI + 2$

б) $SI = SI - 2$

Аналогично загрузка двойных слов и с **EAX** **LODSD**

Общая команда

LODS <Источник>

Функция: то же, что и команды выше, но в качестве источника используется операнд:

а) **~LODSB**, если <Источник> - **BYTE**

б) **~LODSW**, если <Источник> - **WORD**

в) **~LODSD**, если <Источник> - **DWORD**

Примечание: еще есть серия команд **STOSx**, которые отличаются от **LODSx** тем, что они кладут ИЗ **AX** в ПРИЕМНИК (**ES:DI**).

36. Команды сохранения строк.

Команды сохранения строк **STOSB**, **STOSW**, **STOS**, **STOSD**.

с суффиксом **B** - для обработки байтов,

с суффиксом **W** - для обработки слов,

с суффиксом **D** - для обработки двойных слов,

без суффикса - обработка байтов или слов определяется типом операндов.

37. Листинг программы.

Файл с исходным кодом программы, в котором развернуты все макросы, метки заменены на адреса, константы — на их значения. Генерируется ассемблером в процессе подготовки к трансляции. Для получения у **MASM** можно указать флаги **/ Zi / L**, у **ML** — флаг **/ Fl**.

38. Макросредства.

При написании любой программы на ассемблере, возникают некоторые трудности: повторяемость некоторых идентичных или незначительно отличающихся участков программы; необходимость включения в каждую программу участков кода,

которые уже были использованы в других программах; ограниченность набора команд. Для решения этих проблем в языке ассемблер существуют макросредства.

Отличие макросредств от подпрограмм

- если в программе много макровыводов, то увеличивается размер кода
- параметры при записи макрокоманды должны быть известны уже на стадии ассемблирования
- в макрокомандах можно пропускать параметры
- подпрограммы требуют подготовки для вызова (это бывает дольше, чем сам код)

39. Макроопределения (макрофункций и макропроцедур) и макрокоманды.

1. **МАКРООПРЕДЕЛЕНИЕ** - специальным образом оформленная последовательность предложений языка ассемблера, под управлением которой ассемблер (точнее, его часть, называемая **МАКРОГЕНЕРАТОРОМ** или **ПРЕПРОЦЕССОРОМ**) порождает макрорасширения макрокоманд.

2. **МАКРОРАСШИРЕНИЕ** - последовательность предложений языка ассемблера (обыкновенных директив и команд), порождаемая макрогенератором при обработке макрокоманды под управлением макроопределения и вставляемая в исходный текст программы вместо макрокоманды.

3. **МАКРОКОМАНДА** (или **МАКРОВЫЗОВ**) - предложение в исходном тексте программы, которое воспринимается макрогенератором как команда (приказ), предписывающая построить макрорасширение и вставить его на ее место.

В макрокоманде могут присутствовать параметры, если они были описаны в макроопределении.

Макроопределение без параметров однозначно определяет текст макрорасширения. Макроопределение с параметрами описывает множество (возможно, очень большое) возможных макрорасширений, а параметры, указанные в макрокоманде, сужают это множество до одного единственного макрорасширения.

МО делят на однострочные и многострочные. Однострочные МО создаются в основном директивами = (равно), EQU (эквивалентно), TEXTEQU (эквивалентно тексту).

Многострочные МО делят на макропроцедуры (МП) и макрофункции (МФ). Они имеют много общего в собственно определении, но принципиально различаются в конструкциях макрокоманд и в использовании директивы EXTERN.

1.1 Структура макроопределений

ИмяМО MACRO СписФормПар - заголовок МО

.....\

предложения языка ассемблера > - тело МО

...../

ENDM - конец текста МО

ИмяМО - имя макроопределения (МП и МФ),

СписФормПар - имена через запятую, используемые в предложениях тела макроопределения. СписФорПар может отсутствовать.

Завершение работы МФ должно происходить по директиве EXITM с параметром (значение которого будет представлять результат макрофункции – число или текст), а МП – при достижении директивы ENDM, или по директиве EXITM без параметра.

1.2 Формат макрокоманды макропроцедуры (КМП)

ИмяМП СписФактПар

Формат макрокоманды макрофункции (КМФ)

ИмяМФ(СписФактПар)

ИмяМО - имя МО, к которому происходит обращение,

СписФактПар - список фактических параметров через запятую, заменяющих при построении макрорасширения соответствующие формальные параметры.

Элементами списка могут быть имена, строки, выражения.

40. Директива INCLUDE и LOCAL.

INCLUDE <имя файла без кавычек>

Вставляет содержимое файла в код (прямо как в сях) <имяфайлабезкавычек>—ссылка на файл

LOCAL v1, ..., vk (k >= 1)

Указывается, какие имена меток следует рассматривать как локальные.

В макрокомандах:

LOCAL идентификатор[, идентификатор].

В процедурах:

LOCAL элемент[, элемент].[= идентификатор]

M MACRO

...

L:

...

ENDM

Если макрос будет вызван несколько раз, то в коде появятся несколько меток L, что недопустимо. LOCAL L заставляет макрогенератор заменять метки L на имена вида ??xxxx, xxxx 4-значное hex число. [??0000 - ??FFFF]. Макрогенератор запоминает номер, который он использовал последний раз при подстановке (??n) и в следующий раз подставит n+1 (??(n+1))

41. Рекурсия в макроопределениях.

В теле макроопределения может содержаться вызов другого макроопределения или даже того же самого.

RECUR MACRO P

```
MOV AX, P
IF P
    RECUR %P-1
ENDIF
ENDM
```

Такой макрос будет вызывать сам себя, пока P не станет равно 0.

42. Параметры в макросах.

```
ИМЯ MACRO [форм.пар.1[,форм.пар.N]]
    <предложения языка Ассемблер (тело)>
ENDM
```

Фактические параметры перечисляются через запятую, список параметров не обязательно должен присутствовать. Запятые писать обязательно, даже в том случае, если 1 из параметров отсутствует. Так же в качестве параметров можно использовать выражения.

Число фактических параметров может отличаться от формальных параметров в макрокоманде. Если фактических больше, чем формальных, то лишние фактические параметры игнорируются. Если фактических меньше, чем формальных, то формальные параметры, которые не соответствуют фактическим, заменяются на пустую строку.

43. Директивы условного ассемблирования и связанные с ними конструкции.

IF выр блок внутри будет включен в расширение, если выражение != 0

IFE выр блок будет включен в расширение, если выражение = 0

IF1 блок будет включен в расширение при 1 проходе ассемблера.

IF2 блок будет включен в расширение при 2 проходе ассемблера.

Если в качестве имени задана ссылка вперед, она считается неопределенной на 1м проходе и определенной на 2м.

Работают так же, как и обычный IF, но:

IFDEF name

истинно (блок внутри будет в расширении), если имя name было объявлено выше,

IFNDEF name

истинно, если name не было объявлено выше. Также можно использовать с ELSE, как и обычный IF.

44. Директивы IFB и IFNB в макроопределениях.

IFB arg

Если аргумент не задан, то условие является истинно. Можно сказать, что данная директива проверяет значение аргумента на равенство пустой строке. Угловые скобки (<>) должны указываться обязательно

IFNB arg

Действие IFNB обратно IFB. Если аргумент задан, то условие истинно.

45. Директивы IFIDN и IFDIF в макроопределениях.

IFIDN, — истинно, если строки(любой текст) S1 и S2 совпадают. Угловые скобки (<>) должны указываться обязательно

IFDIF , — истинно, если строки S1 и S2 различаются. (if DIFferent)

46. Операции ;: % & < > ! в макроопределениях.

;; — комментарий.

формат: ;; text

% — если требуется **вычисление в строке некоторого константного выражения** или передача его по значению в макрос.

формат: %ВЫРАЖЕНИЕ, например: K EQU 5, %K+2 ; =>7

! — **символ**, идущий после данного знака будет распознан, как символ, а не как операция или директива.

формат: !с, например, если нам надо задать строку &A&: “ !&A!&”

&— **склейка текста** (склеить к параметру). Используется для задания модифицируемых идентификаторов и кодов операций.

формат: &par | &par& | par&, например "Something &A&: &B& something" в этой строке &A& и &B& будут заменены на фактические значения A и B.

< > — **содержит часть текста программы** (в макрорасширении заменяется ровно на то что в скобках). Данный оператор позволяет сгруппировать один или несколько символов в единый строковый литерал. При его обработке препроцессор воспринимает заключенный в угловые скобки текст как один элемент и не интерпретирует его содержимое, например, не разбивает список аргументов, разделенных запятой, на отдельные элементы.

47. Блоки повторения REPT, IRP/FOR, IRPC/FORC, WHILE.

WHILE константное_выражение

...

ENDM

Позволяет повторить некоторый блок операторов в зависимости от значения указанного в ней логического выражения. При выполнении этих директив макрогенератор будет вставлять в макрорасширение указанное количество строк, пока константное выражение не станет равно 0.

REPT константное_выражение

...

ENDM

Предназначена для повторения некоторого блока операторов заданное количество раз. Отличие – REPT автоматически каждую итерацию уменьшает константное выражение на 1, а в WHILE его нужно изменять вручную, но не обязательно на 1.

IRP **формальный_аргумент**, <строка1,строка2>

...

ENDM

Повторяет блок операторов столько раз, сколько в списке в угловых скобках строк. На каждой итерации формальный аргумент принимает значение одной из строк, что дает возможность через форм. арг. к ней обратиться, пока список не исчерпается.

IRPC **формальный_аргумент**,<строка>

...

ENDM

Принцип работы аналогичен IRP, но вместо списка строк передается одна строка и каждую итерацию формальному аргументу присваивается значение одного ее символа.

48. Директива EQU и = в MASM.

<ИМЯ_ИДЕНТИФИКАТОРА> EQU <ВЫРАЖЕНИЕ>:

Эта директива используется для назначения символического имени целочисленному выражению или произвольной текстовой строке. Существует три формата директивы EQU:

имя EQU выражение

имя EQU символ

имя EQU <текст>

В первом случае значение выражения должно иметь целый тип и находиться в допустимых пределах. Во втором случае символ должен быть определен ранее с помощью директивы присваивания (=) или другой директивы EQU. В третьем случае между угловыми скобками <...> может находиться произвольный текст. Если после определения символа с указанным именем оно встретится компилятору в программе, то вместо этого символа будет подставлено соответствующее ему целочисленное значение или текст.

<ИМЯ_ИДЕНТИФИКАТОРА> = <ЧИСЛОВОЕ_ВЫРАЖЕНИЕ>: то же, что и EQU, но константы, заданные =, можно переопределять позднее в коде и работает он только для числовых выражений. При этом числовое выражение может включать что угодно, что может в числовом виде вычислить компилятор, н-р: **3 + 2**, **ASS + 4** (если ASS — заданная выше числовая константа).

49. Директива TEXTEQU в MASM32.

Эта директива впервые появилась в шестой версии MASM. По сути, она очень похожа на директиву EQU и создает так называемый текстовый макрос (text macro). Существует три формата директивы textequ:

имя TEXT EQU <текст>

имя TEXT EQU текстовый__макрое

имя TEXT EQU %константное_выражение

В первом случае символу присваивается указанная в угловых скобках <...>

текстовая строка. Во втором случае — значение заранее определенного текстового макрое. В третьем случае — символической константе присваивается значение целочисленного выражения.

Символ, определенный с помощью директивы TEXT EQU, можно переопределить в программе в любой момент. Этим она отличается от директивы EQU.

50. Типы макроданных text и number.

MacroConstant EQU 123. ;; Числовая макроеконстанта **number**

MacroVar = 123. ;; Числовая макроепеременная **number**

MacroText EQU <string>; Строковая макроепеременная **text**

MacroText TEXT EQU <string>; Строковая макроепеременная **text**

51. Именованные макроеконстанты MASM32

d EQU 0 ;определение макроеконстанты со значением 0

52. Макроеимена, числовые и текстовые макроеконстанты.

d EQU 0 ;определение макроеконстанты со значением 0

MacroText EQU <string> ;; строковая макроепеременная **text**

53. Директивы echo и %echo

Number equ 100

echo Number ; out: Number

%echo Number ; out: 100

Директива echo выводит следующую после нее и пробела часть строки на экран (в поток stdout) , директива %echo вычисляет свой параметр и выводит результат

54. Способы вывода значений макроепеременных и макроеконстант с пояснениями

"display" во время ассемблирования показывает сообщение. За ней должны следовать строка в кавычках или значения байтов, разделенные запятыми.

Директива может быть использована для показа значений некоторых констант, например:

bits = 16

display 'Current offset is 0x'

```

repeat bits/4
  d = '0' + $ shr (bits-%*4) and 0Fh
  if d > '9'
    d = d + 'A'-'9'-1
  end if
  display d
end repeat
display 13,10

```

55. Операции в выражениях, вычисляемых препроцессором MASM:

- Математические операции
+, -, *, /
- Логические

Команда	Описание
AND	Выполняет операцию логического И между двумя операндами
OR	Выполняет операцию логического ИЛИ между двумя операндами
XOR	Выполняет операцию исключающего ИЛИ между двумя операндами
NOT	Выполняет операцию логического отрицание (НЕ) единственного операнда
TEST	Выполняет операцию логического И между двумя операндами, устанавливает соответствующие флаги состояния процессора, но результат операции не записывается вместо операнда получателя данных

- Сдвига

Мнемоника	Описание		Мнемоника	Описание
SHL/SHR	Логический сдвиг		ROR/ROL	Циклический сдвиг
SAL/SAR	Арифметический сдвиг		RCL/RCR	ROR/ROL с установкой CF

56. Подготовка ассемблерных объектных модулей средствами командной строки для использования в средах разработки консольных приложений на ЯВУ.

Masm.exe [flags] prog.asm,,;

Команда получения объектного модуля для последующего использования

57. Добавление ассемблерных модулей в проект консольного приложения С.

В вызывающем модуле сpp должен быть прописан прототип функции и указано соглашение по которым происходит вызов

c++:

```
extern "C" void func(x, y);
```

asm:

```
public func
```

58. Добавление ассемблерных модулей в проект консольного приложения PASCAL.

Мы не делали(

59. Использование ассемблерных вставок в модулях на ЯВУ.

```
__asm {...}
```

60. Вызов подпрограммы С из ассемблерной в VS C++.

.386

```
.model flat
```

```
Public _CallFunc
```

```
extern _printf : proc
```

```
.data
```

```
myint dd 1234
```

```
mystring db 'Это число -> %d <- должно быть 1234',10,0
```

```
.code
```

```
_CallFunc proc
```

```
push offset myint
```

```
push offset mystring
```

```
call _printf
```

```
add esp,12; <-- удаляем из стека аргументы сами
```

```
ret
```

```
_SumFunc endp
```

```
End
```

61. Передача глобальных данных, определённых в консольной программе VS C++, в ассемблерный модуль.

c++:

```
public "C" x;  
asm:  
extrn x
```

62. Передача глобальных данных, определённых в ассемблерном модуле в консольный модуль C.

```
c++:  
extern "C" void func(x, y);  
asm:  
public func
```

63. Средства отладки в CodeView. Примеры.

Подготовка исполняемого файла для отладки в CV

MASM /Zi ИмяФайла.ASM,;

LINK /CO ИмяФайла.OBJ,;;

Загрузка исполняемого файла в CV

CV ИмяФайла.exe [параметры через пробел]

Управление процессом отладки

Q – вывод из отладчика

T[счётчик] - трассировка с заходом в подпрограмму: выполнить одну команду (см. пункт меню F8=Trace) или заданное параметром **счётчик** количество команд

R[счётчик] - трассировка без захода в подпрограмму: выполнить одну команду (см. F10) или заданное параметром **счётчик** количество команд

G[Адрес] – выполнить программу до конца/до точки останова (см. F5 и пункт меню Run/Start) или до команды, заданной параметром **Адрес**

E – выполнить программу медленно до конца или до нажатия клавиши (см. F5 и пункт меню Run/Execute)

L[Параметры] – повторная загрузка программы (параметры программы через пробелы)

[T]>[>]ИмяФайла – перенаправить вывод отладчика и вводимых команд в файл (если задать и **T**, то останется вывод и в окно диалога)

<ИмяФайла –направить команды в отладчик из файла

"(двойная кавычка)– приостановить выполнение команд из файла до нажатия любой клавиши

***** – начало комментария в отдельной строке командного файла

: – приостановить на 0,5 секунды выполнение очередной команды

. – отобразить текущую команду (она выделена синим фоном) в окне просмотра

Модификация и отображение данных в окне диалога данных

? Выражение[,Формат] – вычислить выражение

D[Тип] [Адрес | Диапазон] – вывод данных (по умолчанию тип – ранее использованный в командах **D|TP|W**)

E[Тип] Адрес [список] – вычисление выражений списка, преобразование к

заданному типу и размещение в ОП, начиная с заданного адреса.

E Адрес список – вычисление выражений списка и ввод в диалоге по типу, определённом в предыдущих командах **E W TP**.

F Диапазон список – циклическая запись в ОП в указанный диапазон данных из списка

M Диапазон Адрес – пересылка данных из диапазона в ОП указанному адресу

R[ИмяРегистра] [[=] [Выражение] – присвоить регистру AX | BX | CX | DX | SI | DI | CS | DS | ES | SS | BP | IP | F-регистру флагов.

R ИмяРегистра – изменить значение регистра AX | BX | CX | DX | SI | DI | CS | DS | ES | SS | BP | IP | F в диалоге.

Точки безусловного останова

BP[Адрес] [КолПрох] [”командыCV”] – установить точку безусловного останова на команде по заданному параметром **Адрес** адресу с остановкой перед выполнением команды перед последним из заданного **КолПрох** числа проходов через неё

и выполнить команды, перечисленные через ; в списке **”командыCV”**.

BL– вывод в окно диалога построчно информации о всех точках безусловного останова: - номер точки (0, 1,...), - знак активности точки (e – активна, d – временно запрещена), адрес точки (сегмент:смещение), - **КолПрох**, - **”командыCV”**.

BC *| список – удалить все точки безусловного останова (*), или только указанные в параметре **список** через пробелы

BD *| список – временно запретить все точки безусловного останова (*) или только указанные в параметре **список** через пробелы.

BE *| список – разрешить все временно запрещенные точки безусловного останова (*) или только указанные в параметре **список** через пробелы.

Операторы наблюдения и точки условного останова

W[Тип] ДиапазонАдресов – установить в окне наблюдения оператор наблюдения - строку, отображающую текущие значения в ячейках указанного диапазона как данных заданного типа.

W? Выражение[,Формат] – установить в окне наблюдения *оператор наблюдения* - строку, отображающую текущие значение выражения в заданном формате.

WP? Выражение[,Формат] – установить в окне наблюдения *точку наблюдения* - строку, отображающую условие останова: при истинности (не 0) выражения

TP? Выражение[,Формат] – установить в окне наблюдения *точку трассировки* - строку, отображающую условие останова: при изменении выражения

TP[Тип] ДиапазонАдресов – установить в окне наблюдения *точку трассировки* - строку, отображающую условие останова: при изменении значения в заданном диапазоне

W – отобразить в окне диалога информацию по всем операторам наблюдения и точкам останова.

Y *| список – удалить все операторы наблюдения и точки условного останова (*), или только указанные в параметре **список** через пробелы.

64. Средства отладки в средах разработки консольных приложений на ЯВУ.

Можно установить точки останова, посмотреть дизассемблированный код.

65. Способы адресации. Термины и смысл.

В инструкциях программ операнды строятся из объектов, представляемых собой имена переменных и меток, представленных своим сегментом смещения (регистры, числа, символические имена с численным значением). Запись этих объектов с использованием символов: [], +, -, . в различных комбинациях называют способами адресации.

1. **Непосредственная адресация (значение):** операнд задается непосредственно в инструкции и после трансляции входит в команду, как составная часть 1 и 2 байта.

```
MOV AL,2 ;(1байт)
```

```
MOV BX,0FFFFH ;(2байта)
```

2. **Регистровая адресация:** значение операнда находится в регистре, который указывается в качестве аргумента инструкции.

```
MOV AX,BX
```

3. **Прямая адресация:** в инструкции используется имя переменной, значение которой является операндом.

```
MOV AX,X ; в AX значение X
```

```
MOV AX,CS:Y ; в AX CS по смещению Y
```

4. **Косвенная регистровая адресация:** в регистре, указанном в инструкции, хранится смещение (в сегменте) переменной, значение которой и является операндом. Имя регистра записывается в [].

```
MOV BX, OFFSET Z
```

```
...
```

```
MOV BYTE PTR[BX], 'ASS' ;в байт, адресованный регистром BX, запомнить 'ASS' в сегменте DS
```

```
MOV AX, [BX] ; в AX запомнить слово из сегмента, адрес BX со смещением DS
```

```
MOV BX, CS:[SI] ; записать данные в BX, находящиеся в CS по смещению SI
```

5. **Косвенная базово-индексная адресация (адресация с индексацией и базированием):** смещение ячейки памяти, где хранятся значения операндов, вычисляется, как сумма значений регистров, записанных в качестве параметра в инструкции. $[R1+R2]$, где $R1 \in \{BX, BP\}$, $R2 \in \{SI, DI\}$

```
MOV AX,[BX + DI]
```

```
MOV AX,[BP + SI]
```

Для BX по умолчанию сегменты из регистра DS. Для BP по умолчанию сегменты из регистра SS.

6. **[Косвенная] базово-индексная адресация со смещением :** (прямая с базированием и индексированием), отличается от п.5 тем, что ещё прибавляется смещение, которое может быть представлено переменной или ЦБЗ.

```
;[R1 + R2 ± ЦБЗ]
```

```
;[R1][R2 ± ЦБЗ]
```

```
;имя[R1 + R2 ± Абс.выражение]
```

```
;имя[R1][R2 ± Абс.выражение]
```

```
MOV AX, ARR[EBX][ECX * 2]
```

```
;ARR + (EBX) + (ECX) * 2
```

```
; вот такой монстр
; [Имя][База][Индекс [* масштаб]][Абс. выражение]
; Смещение операнда = Смещение имени + значение базы + значение индекса *
масштаб + значение абсолютного выражения
```

66. Связывание подпрограмм. Конвенции C, PASCAL, STDCALL, РЕГИСТРОВАЯ.

Связывание подпрограмм — совокупность действий по:

- передаче управления подпрограмме
- передаче параметров через регистры, общую область памяти, области параметров
- возврату управления из подпрограмм
- запоминанию и восстановлению состояния вызывающей программы

Все это — связывание подпрограмм в исполняемом модуле. В более широком плане, когда речь идет о динамическом вызове подпрограмм или программ, хранящиеся на диске в виде отдельных модулей, к этому перечню добавляется:

1. загрузка модуля подпрограмм в ОЗУ и разрешение внешних ссылок (по передаче управления и по передаче данных).
2. освобождение памяти, после завершения работы вызываемого модуля и возврат управления из вызываемого модуля.

Конвенция Pascal заключается в том, что параметры из программы на языке высокого уровня передаются в стеке и возвращаются в регистре AX/EAX, поместить параметры в стек в естественном порядке. За уничтожение параметров в стеке отвечает вызываемая программа.

Конвенция C. Параметры помещаются в стек в обратном порядке, и, в противоположность PASCAL-конвенции, удаление параметров из стека выполняет вызывающая процедура.

Тип возвращаемого значения	Регистр
<code>unsigned char</code>	al
<code>char</code>	al
<code>unsigned short</code>	ax
<code>short</code>	ax
<code>unsigned int</code>	eax
<code>int</code>	eax
<code>unsigned long int</code>	edx:eax
<code>long int</code>	edx:eax

Существует конвенция передачи параметров **STDCALL**, отличающаяся и от C, и от PASCAL-конвенций, которая применяется для всех системных функций Win32 API. Здесь параметры помещаются в стек в обратном порядке, как в C, но процедуры должны очищать стек сами, как в PASCAL.

Регистровое соглашение FASTCALL. В этом случае параметры в функции также передаются по возможности через регистры. Например, при вызове функции с шестью параметрами
some_proc(a,b,c,d,e,f);
первые три параметра передаются соответственно в EAX, EDX, ECX, а только начиная с четвертого, параметры помещают в стек в обычном обратном порядке. В случае если стек был задействован, освобождение его возлагается на вызываемую процедуру.

67. Многострочные макроопределения. Формальные и фактические параметры.

Многострочные МО делят на макропроцедуры (МП) и макрофункции (МФ). Они имеют много общего в собственно определении, но принципиально различаются в конструкциях макрокоманд и в использовании директивы EXTERN.

Структура макроопределений

ИмяМО MACRO СписФормПар - заголовок МО

.....

предложения языка ассемблера > - тело МО

.....

ENDM - конец текста МО

ИмяМО - имя макроопределения (МП и МФ),

СписФормПар - имена через запятую, используемые в предложениях тела макроопределения. СпсФормПар может отсутствовать.

Завершение работы МФ должно происходить по директиве EXITM с параметром (значение которого будет представлять результат макрофункции – число или текст), а МП –при достижении директивы ENDM, или по директиве EXITM без параметра.

Формат макрокоманды макропроцедуры (КМП)

ИмяМП СпсФактПар

Формат макрокоманды макрофункции (КМФ)

ИмяМФ(СпсФактПар)

ИмяМО - имя МО, к которому происходит обращение,

СпсФактПар - список фактических параметров через запятую, заменяющих при построении макрорасширения соответствующие формальные параметры.

Элементами списка могут быть имена, строки, выражения.

68. Многострочные макроопределения и директивы условного ассемблирования

Структуры условного ассемблирования IF W, где W, W1, W2,... вычисляемые препроцессором выражения:

IF W Директива IF условного ассемблирования

..... \ Блок предложений ассемблера, выполняемых

..... / при истинности условия

ENDIF Директива условного ассемблирования

IF W1 Директива IF условного ассемблирования

..... \ Блок1 предложений ассемблера, выполняемых

..... / при истинности условия

ELSEIF W2 Директива условного ассемблирования

..... \ Блок2 предложений ассемблера, выполняемых

..... / , если условие ложно

ELSE Директива условного ассемблирования

..... \ Блок3 предложений ассемблера, выполняемых

..... / , если условие ложно

ENDIF Директива условного ассемблирования

Блоков ELSEIF может быть несколько или не быть совсем.

Другие директивы IF условного ассемблирования:

IFB <par> - условие истинно, если фактический параметр par
не был задан в МКоманде (скобки <> обязательны)

IFNB <par> - условие истинно, если фактический параметр par
был задан в МКоманде (скобки <> обязательны)

IFIDN <s1>,<s2> - условие истинно, если строки s1 и s2
совпадают (скобки <> обязательны)

IFDIF <s1>,<s2> - условие истинно, если строки s1 и s2
различаются (скобки <> обязательны)

IF www - условие истинно, если значение www<>0

IFE www - условие истинно, если значение www=0

IFDEF nam - условие истинно, если имя nam было описано выше

IFNDEF nam - условие истинно, если имя nam не было описано выше

IF1 - условие истинно 1-м шаге ассемблирования

IF2 - условие истинно 2-м шаге ассемблирования

69. Многострочные макроопределения и директивы генерации ошибок

<u>Директива</u>	<u>№ ошибки</u>	<u>Текст сообщения на экране</u>
.ERRB <s>	94	Forced error - пустая строка
.ERRNB <s>	95	Forced error - непустая строка
.ERRIDN <s1>,<s2>	96	Forced error - строки одинаковые
.ERRDIF <s1>,<s2>	97	Forced error - строки разные
.ERRE www	90	Forced error - выражение = 0
.ERRNZ www	91	Forced error - выражение <> 0
.ERRNDEF nam	92	Forced error - имя не определено
.ERRDEF nam	93	Forced error - имя определено
.ERR1	87	Forced error - pass 1
.ERR2	88	Forced error - pass 2
.ERR	89	Forced error

70. Многострочные макроопределения и операции в них.

;; — комментарий.

формат: `;; text`

% — если требуется **вычисление в строке некоторого константного выражения** или передача его по значению в макрос.

формат: `%ВЫРАЖЕНИЕ`, например: `K EQU 5, %K+2 ; =>7`

! — символ, идущий после данного знака будет распознан, как символ, а не как операция или директива.

формат: `!с`, например, если нам надо задать строку `&A&`: `“ !&A!& ”`

& — склейка текста (склеить к параметру). Используется для задания модифицируемых идентификаторов и кодов операций.

формат: `&par | &par& | par&`, например `"Something &A&: &B& something"` в этой строке `&A&` и `&B&` будут заменены на фактические значения А и В.

<> — содержит часть текста программы (в макрорасширении заменяется ровно на то что в скобках). Данный оператор позволяет сгруппировать один или несколько символов в единый строковый литерал. При его обработке препроцессор воспринимает заключенный в угловые скобки текст как один элемент и не интерпретирует его содержимое, например, не разбивает список аргументов, разделенных запятой, на отдельные элементы.

71. Макропроцедуры. Определения и вызовы

Структура макроопределений

ИмяМО MACRO СписФормПар - заголовок МО

.....\

предложения языка ассемблера > - тело МО

...../

ENDM - конец текста МО

где

ИмяМО - имя макроопределения (МП и МФ),

СписФормПар - имена через запятую, используемые в предложениях тела макроопределения. СписФормПар может отсутствовать.

Завершение работы МП должно происходить при достижении директивы ENDM, или по директиве EXITM без параметра

72. Рекурсивные макропроцедуры

Пример макропроцедуры, вычисляющей $S=P!$.

MP_REC MACRO P,S

S=P

MOV EAX,P


```

    IFE P EQ 1
        S=1
    ELSE
        MP_REC P-1
        S=S*P
    ENDIF
    ENDM
.CODE
    MP_REC 3,S
    MOV EAX,S

```

В листинге мы видим, что на каждом шаге рекурсии в макропеременной P будет меняться **текст** выражения, но значение вычисляться не будет

73. Макрофункции. Определения и вызовы

Структура макроопределений

```

ИмяМО MACRO СписФормПар    - заголовок МО
.....\
предложения языка ассемблера > - тело МО
...../
    ENDM                    - конец текста МО

```

где

ИмяМО - имя макроопределения (МП и МФ),

СписФормПар - имена через запятую, используемые в предложениях тела макроопределения. СписФормПар может отсутствовать.

Завершение работы МФ должно происходить по директиве EXITM с параметром (значение которого будет представлять результат макрофункции – число или текст),

74. Рекурсивные макроопределения

В теле макроопределения может содержаться вызов другого макроопределения или даже того же самого.

```

RECUR MACRO P
    MOV AX, P
    IF P
        RECUR %P-1
    ENDIF
ENDM

```

Такой макрос будет вызывать сам себя, пока P не станет равно 0. Например, RECUR развернется в

```

MOV AX, 3
MOV AX, 2
...

```

75. Числовые макроконстанты. Определение и примеры использования

MacroConstant EQU 123 ;; Числовая макроконстанта **number**

76. Числовые макропеременные. Определение и примеры использования

MacroVar = 123 ;; Числовая макропеременная **number**

Целочисленная макропеременная. Имеет тип INT (dword). Может участвовать во всех арифметических выражениях MASM. Как переменная она может изменять своё значение.

Макроконстанта может иметь целочисленное значение. Её значение не может быть повторно изменено.

77. Константные выражения. Операции, вычисляемые препроцессором в выражениях

Константное выражение состоит из комбинации цифровых литералов, операторов и определенных символьных констант. Значение константного выражения определяется во время трансляции программы и не может меняться во время выполнения программы. Ниже приведено несколько примеров константных выражений, включающих только цифровые литералы:

- 5;
- 26,5;
- 4 * 20;
- -3 * 4/6;
- -2,301E+04.

78. Текстовые макропеременные. Способы определения

MacroText EQU <string> ;; строковая макропеременная **text**

Текстовой макро может быть любой строкой не более 255 символов. Поскольку он имеет статус переменной, его значение может быть изменено.

79. Блоки повторения REPT, FOR, FORC, WHILE.

REPT константное_выражение

...

ENDM

Предназначена для повторения некоторого блока операторов заданное количество раз. Отличие – REPT автоматически каждую итерацию уменьшает константное выражение на 1, а в WHILE его нужно изменять вручную, но не обязательно на 1.

FOR формальный_аргумент, <строка1,строка2>

...

ENDM

Повторяет блок операторов столько раз, сколько в списке в угловых скобках строк. На каждой итерации формальный аргумент принимает значение одной из строк, что дает возможность через форм. арг. к ней обратиться, пока список не исчерпается.

FORC формальный_аргумент,<строка>

...

ENDM

Принцип работы аналогичен IRP, но вместо списка строк передается одна строка и каждую итерацию формальному аргументу присваивается значение одного ее символа.

WHILE константное_выражение

...

ENDM

Позволяет повторить некоторый блок операторов в зависимости от значения указанного в ней логического выражения. При выполнении этих директив макрогенератор будет вставлять в макрорасширение указанное количество строк, пока константное выражение не станет равно 0.

80. Стандартные макрофункции обработки строк @CATSTR и @SUBSTR.

Макрос функция, которая объединяет одну или несколько строк. Возвращает строку.

@CatStr(string1 [[, string2...]])

Макрос функция, которая возвращает подстроку начиная с *позиции*.

@SubStr(string, позиции [[, длина]])

81. Инструменты отладки макросредств. Окно командной строки, листинг, сообщения препроцессора

1. **Окно командной строки** (директивы вывода сообщений препроцессора ECHO и %ECHO)

2. **Листинг программы** (машинные команды и макрорасширения макросов)

Файл с исходным кодом программы, в котором развернуты все макросы, метки заменены на адреса, константы — на их значения. Генерируется ассемблером в процессе подготовки к трансляции. Для получения у MASM можно указать флаги / Zi /L, у ML — флаг /Fl.

3. **Директивы управления листингом** (.LALL, .XALL .SALL, .NOLIST в MASM16 и те же или подобные в MASM32: .LISTALL, .LISTMACROALL) и составом макроопределений (INCLUDE, PURGE).

INCLUDE ФБиб

Функция: Включает в ассемблируемый текст текст из файла ФБиб на место директивы INCLUDE.

PURGE ИмяМО[,ИмяМО..]

Функция: Удаляет из обработки макроопределения, имена которых перечислены в качестве параметров.

Цель - экономия памяти при работе ассемблера.

Директива .LALL: далее включать в листинг программы полные макрорасширения (кроме комментариев после ;:).

Директива .XALL (.LISTMACROALL): далее включать в листинг программы только те предложения макрорасширений, которые генерируют коды и данные.

.SALL: далее не выводить тексты макрорасширений в листинг.

.NOLIST: далее вообще прекратить вывод листинга до появления иной директивы.

82. Применение макросредств при определении переменных

Определение переменных A0,A1,A2,A3 с начальными значениями 0,1,2,3 соответственно.

```
IRP X,<0,1,2,3> ;параметры - числа
    A&X DB X
ENDM
```

Описание переменных Z_AAAA, Z_BBBB, Z_CCCC с начальными значениями 'AAAA', 'BBBB', 'CCCC' соответственно.

```
DCL MACRO Y
    IRP X,<AAAA,BBBB,CCCC> ;параметры - строки символов
        Y&&X DB '&X&'
    ENDM
ENDM
```

83. Применение макросов при создании фрагментов кода

```
PUSH_ALL MACRO LIST
    IRP REG, <LIST> ;; for REG in LIST
        IFIDN <REG>, <F> ;; if REG == F
            PUSHF
        ELSE
            PUSH REG
        ENDIF
    ENDM
ENDM

POP_ALL MACRO LIST
    LOCAL count
    count = 0
    FOR param, <LIST> ;; создаем переменные varXX
        count = count + 1
        @CatStr(var, %count) TEXTEQU <param>
    ENDM
    REPT count
        pop @CatStr(var, %count)
        count = count - 1
    ENDM
ENDM
```

```
PUSH_ALL <ESI, EAX>
```

перейдет

```
PUSH ESI
```

```
PUSH EAX
```

```
POP_ALL <ESI, EAX>
```

перейдет

```
POP EAX
```

```
POP ESI
```