

Оглавление

1) Архитектура МП 8088 и 80386.....	3
2) Характеристики регистров.	3
3) Флаги.....	5
4) Сегментные регистры по умолчанию.	7
5) Образование физического адреса.	8
6) Сегментный префикс.	8
7) Структура программы одномодульной MS DOS. Повторные описания сегментов.	8
8) Возможные структуры кодового сегмента.	9
9) Возможные способы начала выполнения и завершения программы MS DOS типа .exe.	10
10) Структура программы из нескольких исходных модулей MS DOS.	10
11) Переменные, метки, символические имена и их атрибуты.	11
12) Виды предложений языка Ассемблер.	12
13) Директивы (псевдооператоры): назначение и формы записи.	12
14) Стандартные директивы описания сегментов: формат записи заголовков директив и назначение параметров.....	13
15) Возможные комбинации сегментов и умолчания. !!!	14
16) Директива ASSUME.	14
17) Структура процедур.	14
18) Директива END.	15
19) Внешние имена.....	15
20) Типы данных и задание начальных значений.	15
21) Способы описания меток, типы меток.(типы меток???)	16
22) Команды условных переходов при работе с ЦБЗ и ЦСЗ.....	17
23) Команды организации циклов.	18
24) Директива ORG.(не очень).....	18
25) Способы адресации.	19
26) Организация рекурсивных подпрограмм.!!!	20
27) Арифметические команды (для ЦБЗ и ЦСЗ)	20
28) Связывание подпрограмм.	21
29) Команда CALL. Использование прямой и косвенной адресации.	22
30) Способы передачи параметров подпрограмм.....	22
31) Способы сохранения и восстановления состояния вызывающей программы (кто выполняет и в чьей памяти).....	23
32) Соглашени о связях в Turbo Pascal, Turbo C, Delphi, VS C++.....	23

33)	Команды сдвига.....	24
34)	Команды логических операций.(из евтмхыча добавить)	24
35)	Команды обработки строк и префиксы повторения.....	25
36)	Команды пересылки строк.	25
37)	Команды сравнения строк.	26
38)	Команды сканирования строк.	27
39)	Команды загрузки строк.....	28
40)	Команды сохранения строк.....	30
41)	Листинг программы.	30
42)	Макросредства.....	30
43)	Описания макроопределений (макрофункций и макропроцедур) и макрокоманд.	30
44)	Директива INCLUDE.....	31
45)	Рекурсия в макроопределениях.....	31
46)	Параметры в макросах.	32
47)	Директива LOCAL.....	32
48)	Директивы условного ассемблирования IF, IFE, IF2, IFIDN/IFIDNI, IFDIF /IFDIFI, IFDEF, IFNDEF и связанные с ними конструкции.	33
49)	Директивы IFB и IFNB в макроопределениях.....	33
50)	Директивы IFIDN и IFDIF в макроопределениях.....	34
51)	Операции ;; % & < > ! в макроопределениях.....	34
52)	Блок повторения REPT.....	34
53)	Блок повторения IRP/FOR.....	35
54)	Блок повторения IRPC/FORC.....	35
55)	Блок повторения WHILE.....	35
56)	Директива EQU в MASM.....	35
57)	Директива TEXTEQU в MASM32.....	36
58)	Директива = в MASM.....	36
59)	Типы макроданных text и number (см листинг) !!!---	37
60)	Именованные макроконстанты MASM32 !!!.....	37
61)	Макроимена, числовые и текстовые макроконстанты - значения. !!!	37
62)	Директивы echo и %echo.....	37
63)	Способы вывода значений макропеременных и макроконстант с пояснениями !!!	37
64)	Операций в выражениях MASM: !!!.....	37
65)	Подготовка ассемблерных объектных модулей средствами командной строки для использования в Delphi и VS C++.	39
66)	Добавление ассемблерных модулей в проект консольного приложения VS C++	39

67)	Добавление ассемблерных модулей в проект консольного приложения Delphi.....	39
68)	Использование ассемблерных вставок в модулях .cpp. !!!.....	39
69)	Вызов из ассемблерной подпрограммы C в VS C++. !!!	39
70)	Передача глобальных данных, определённых в консольной программе VS C++, в ассемблерный модуль. !!!.....	39
71)	Передача глобальных данных, определённых в ассемблерном модуле в консольный модуль .cpp VS C++. !!!	40
72)	Средства отладки в CodeView. Примеры. !!!.....	40
73)	Средства отладки в VS C++. Примеры. !!!	40
74)	Получение дизассемблированного кода в VS C++ !!!	40

1) Архитектура МП 8088 и 80386

8086 — 16разрядный микропроцессор. Имеет 14 16разрядных регистров, 16-разрядную шину данных, 20разрядную шину адреса (поддерживает адресацию до 1 Мб памяти), поддерживает 98 инструкций.

8088 16разрядный микропроцессор. Имеет 14 16разрядных регистров. у него 8-разрядная шина данных.

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

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).

Регистры общего назначения	Аккумулятор	EAX	
		AX	
		AH	AL
		EBX	
	База	BX	
		BH	BL
		ECX	
		CX	
	Счётчик	CH	CL
		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 RRF 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.	...	

Всего архитектуры 8086 и 8088 содержат 14 16битных регистров

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.

Каждый бит в регистре **FLAGS** является флагом. **Флаг** – это один или несколько битов памяти, которые могут принимать двоичные значения (или комбинации значений) и характеризуют состояние какого-либо объекта.

Обычно флаг может принимать одно из двух логических значений. Поскольку в нашем случае речь идёт о бите, то каждый флаг в регистре может принимать либо значение 0, либо значение 1. Флаги устанавливаются в 1 при определённых условиях, или установка флага в 1 изменяет поведение процессора. На рис. 2.4 показано, какие флаги находятся в разрядах регистра FLAGS.

Бит	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Флаг	0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF	

Бит	Обозначение	Название	Описание
0	CF	Carry Flag	Флаг переноса. Устанавливается в 1, если результат предыдущей операции не уместился в приёмнике и произошёл перенос из старшего бита или если требуется заём (при вычитании). Иначе установлен в 0. Например, этот флаг будет установлен при переполнении, рассмотренном в предыдущем разделе .
1	1	-	Зарезервирован.
2	PF	Parity Flag	Флаг чётности. Устанавливается в 1, если младший байт результата предыдущей команды содержит чётное количество битов, равных 1. Если количество единиц в младшем байте нечётное, то этот флаг равен 0.
3	0	-	Зарезервирован.

4	AF	Auxiliary Carry Flag	Вспомогательный флаг переноса (или флаг полупереноса). Устанавливается в 1, если в результате предыдущей операции произошёл перенос (или заём) из третьего бита в четвёртый. Этот флаг используется автоматически командами двоично-десятичной коррекции.
5	0	-	Зарезервирован.
6	ZF	Zero Flag	Флаг нуля . Устанавливается 1, если результат предыдущей команды равен 0.
7	SF	Sign Flag	Флаг знака . Этот флаг всегда равен старшему биту результата.

8	TF	Trap Flag	Флаг трассировки (или флаг ловушки). Он был предусмотрен для работы отладчиков в пошаговом выполнении, которые не используют защищённый режим. Если этот флаг установить в 1, то после выполнения каждой программной команды управление временно передаётся отладчику (вызывается прерывание 1).
9	IF	Interrupt Enable Flag	Флаг разрешения прерываний . Если сбросить этот флаг в 0, то процессор перестанет обрабатывать прерывания от внешних устройств. Обычно его сбрасывают на короткое время для выполнения критических участков программы.

10	DF	Direction Flag	Флаг направления. Контролирует поведение команд обработки строк. Если установлен в 1, то строки обрабатываются в сторону уменьшения адресов, если сброшен в 0, то наоборот.
11	OF	Overflow Flag	Флаг переполнения. Устанавливается в 1, если результат предыдущей арифметической операции над числами со знаком выходит за допустимые для них пределы. Например, если при сложении двух положительных чисел получается число со старшим битом, равным единице, то есть отрицательное. И наоборот.
12 13	IOPL	I/O Privilege Level	Уровень приоритета ввода/вывода.
14	NT	Nested Task	Флаг вложенности задач.
15	0	-	Зарезервирован.

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 `jmp`) 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) Образование физического адреса.

Память представляет собой линейную последовательность байтов, поделенную на параграфы.

Параграфы - последовательности из 16 идущих подряд байт, у первого из которых адрес кратен 16.

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

$$\text{<полный адрес>} = \text{<номер сегмента>} * \text{<размер сегмента>} + \text{<смещение>}$$

6) Сегментный префикс.

Конструкция вида

AS:X

где AS — какойто сегментный регистр (DS, ES, SS, CS), а X — какая-то метка (ближний адрес) или другой регистр (SI, DI, SP, IP). Явно указывает, какой регистр мы используем для получения номера сегмента, к которому мы обращаемся.

Н-р: `MOV AX, ES:X`

7) Структура программы одномодульной MS DOS. Повторные описания сегментов.

Пример одномодульной программы:

```
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 ASS
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
```

8) 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.

Еще вместо метки точки входа можно использовать FAR-процедуру. Тогда можно заканчивать программу точно так же по INT 21h / AH: 4Ch, а можно просто делать в конце RET, перед этим загрузив в стек 0, а в начале программы перед выставлением регистра DS загрузить его старое значение. Остальное не отличается.

9) Возможные способы начала выполнения и завершения программы MS DOS типа .exe.

Способы входа: END <метка в кодовом сегменте>, END <процедура(far)>
Ставится в коде программы после сегмента кода (только ОДИН РАЗ ВО ВСЕЙ ПРОГРАММЕ).

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

При этом если выход с помощью RET, то на вершине 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).

10) Структура программы из нескольких исходных модулей MS DOS.

```
; 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      ;Здесь нихрена нет, поэтому модуль
          ; неглавный~

```

Сначала создаются объектные файлы отдельных модулей с помощью компилятора (masm.exe), а затем всё компоуется в исполняемый файл линковщиком (link.exe).

11) 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 поочередно B,C и D
A DB 123	; Переменная A
B EQU 456	; Константа B
C = 902	; Константа C, которую можно переобъявлять

12) 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 ‘;’.

13) 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[<имясег>]:** задает положение начала сегмента кода
 - **.STACK[<размер>]:** задает размер стека. По дефолту — 10246
 - **.DATA:** начало или продолжение сегменты инициализированных данных

- **.DATA?**: начало или продолжение сегмента инициализированных 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.

<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).

14) Стандартные директивы описания сегментов: формат записи заголовков директив и назначение параметров.

```
<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', ...)

Упрощенные директивы сегментации

- **.CODE[<имясег>]**: задает положение начала сегмента кода
- **.STACK[<размер>]**: задает размер стека. По дефолту — 10246
- **.DATA**: начало или продолжение сегмента инициализированных данных

- **.DATA?**: начало или продолжение сегмента неиниц. данных

15) 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.

(хуйня)

16) Директива 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.

17) 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 строки 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 сегм-ами
    ...
    RET ;тоже дальняя команда
ИМЯ_ПРОЦ ENDP

```

18) Директива 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.

19) 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
...

```

20) 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 "?".

21) Способы описания меток, типы меток.(типы меток???)

Метка в языке ассемблера может содержать следующие символы:

Буквы: от A до Z и от a до z

Цифры: от 0 до 9

Спецсимволы: знак вопроса (?)

точка (.) (только первый символ)

знак "коммерческое эт" (@)

подчеркивание (_)

доллар (\$)

Первым символом в метке должна быть буква или спецсимвол. Цифра не может быть первым символом метки, а символы \$ и ? иногда имеют специальные значения и обычно не рекомендуются к использованию

Если метка располагается перед командой процессора, сразу после нее всегда ставится символ «:» (двоеточие), который указывает ассемблеру, что надо создать переменную с этим именем, содержащую адрес текущей команды:

some_loop:

lodsw ; считать слово из строки,

cmp ax,7 ; если это 7 - выйти из цикла

loopne some_loop

Когда метка стоит перед директивой ассемблера, она обычно оказывается одним из операндов этой директивы и двоеточие не ставится:

codesg segment

lodsw ; считать слово из строки,

cmp ax,7 ; если это 7 - выйти из цикла

codesg ends

22) Команды условных переходов при работе с ЦБЗ и ЦСЗ.

Имеется 19 команд, имеющих 30 мнемонических кодов операций. (по правде, команд и мнемоник дохрена) 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	ЦСЗ	(ZF == 0) && (SF == OF)
JGE JNL	Greather or Equal	ЦСЗ	SF == OF
JL JNGE	Less	ЦСЗ	SF != OF
JLE JNG	Less or Equal	ЦСЗ	(ZF == 1) (SF != OF)
JA JNBE	Above	ЦБЗ	(CF == 0) && (ZF == 0)
JAЕ JNB	Above or Equal	ЦБЗ	CF == 0
JB JNAЕ	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)	ЦСЗ	SF == 1
JNS	Not Sign (>= 0)	ЦСЗ	SF == 0
JP	Parity	все	PF == 1
JNP	Not Parity	все	PF == 0
JCXZ	CX is Zero	все	CX == 0

23) Команды организации циклов.

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

Мнемоника	Описание
<code>LOOP</code>	--CX. Если $(CX \neq 0)$ — повторяем цикл (переходим к метке, указанной в команде)
<code>LOOPE LOOPZ</code>	--CX. Если $(CX \neq 0) \&\& (ZF == 1)$ — повторяем цикл.
<code>LOOPNE LOOPNZ</code>	--CX. Если $(CX \neq 0) \&\& (ZF == 0)$ — повторяем цикл.

Примеры использования:

Обычный цикл:

Ассемблерный код	Аналог на C (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 C (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 запусивания я юзаю // другую переменную for (int CX2 = 7; CX2 > 0; --CX2) BX += 10; }</pre>

24) Директива `ORG.`(не очень)--

предназначена для записи в счетчик адреса сегмента значения своего операнда. То есть при помощи этой директивы можно разместить команду (или данные) в памяти микроконтроллера по любому адресу. Пример использования директивы `ORG` для размещения подпрограмм обработки прерываний на векторах прерываний:

```

reset:
    LJMP main

;Перезагрузка таймера-----
    ORG 0bh ;Вектор прерывания от таймера 0
    LJMP IntT0

;Перезагрузка таймера-----
    ORG 23h ;Вектор прерывания последовательного порта
    LJMP IntSPORT

IntT0:
    mov     TL0,    #LOW(-(F_ZQ/12)*10-2)      ;Настроить таймер
    mov     TH0,    #HIGH(-(F_ZQ/12)*10-2)     ;на период 10мс
    reti

;Начало основной программы микроконтроллера -----
main:
    mov     SP,#VerShSteka      ;Настроить указатель стека на вершину стека
    call    init               ;Настроить микроконтроллер
;-----

```

Необходимо отметить, что при использовании этой директивы возможна ситуация, когда программист приказывает транслятору разместить новый код программы по уже написанному месту, поэтому использование этой директивы допустимо только в крайних случаях. Обычно это использование векторов прерываний.

25) 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 ± 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 а
```

26) Организация рекурсивных подпрограмм.!!!

27) Арифметические команды (для ЦБЗ и ЦСЗ)

ADD Приемник, Источник

Сложение байтов или слов

Функция: Приемник := Приемник + Источник

изменяет флаги: OF, SF, ZF, AF, PF, CF.

ADC Приемник, Источник

ADC Приемник, Источник

Сложение байтов или слов с переносом

Функция: Приемник := Приемник + Источник+CF
изменяет флаги:OF, SF, ZF,AF,PF,CF.

INC Приемник

Увеличение байта или слова на 1

Функция: Приемник:=Приемник+1
изменяет флаги:OF,SF,ZF,AF,PF.
! не влияет на CF !

AAA – коррекция в AL неупакованного кода при сложении

DAA – коррекция в AL упакованного кода после сложения

SUB Приемник, Источник

Вычитание байтов или слов

Функция: Приемник := Приемник-Источник
изменяет флаги:OF, SF, ZF,AF,PF,CF.

SBB Приемник,Источник

Вычитание байтов или слов с заемом

Функция: Приемник:=Приемник-Источник-CF
изменяет флаги:OF, SF, ZF,AF,PF,CF.

DEC Приемник

Уменьшение байта или слова на 1

Функция: Приемник:=Приемник-1
изменяет флаги:OF,SF,ZF,AF,PF.
! не влияет на CF !

NEG Приемник

Изменение знака байта или слова

Функция: Приемник:=-Приемник
изменяет флаги:OF,SF,ZF,AF,PF.
! Если Приемник <> 0, то CF:=1,
иначе CF:=0 и Приемник сохранит значение 0!
! Попытка изменить знак для -128 или -32768
не изменит операнд, но установит OF:=1!

28) 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.

29) Команда CALL. Использование прямой и косвенной адресации.

CALL передает управление в другую строку программы, при этом сохраняя в стеке адрес возврата 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. Передача управления подпрограмм с помощью команды CALL:

CALL имя_процедуры или имя_метки в подпрограмме(прямая адресация)

CALL [переменная] (косвенная адресация)

30) Способы передачи параметров подпрограмм.

1. Через регистры:

плюсы: быстро, просто

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

2. 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

3. Через общий стек (используется чаще всего):

плюсы: можно передавать большое количество параметров

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

4. 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

```
PARSEG SEGMENT COMMON ; потом грузим этот сегмент в DS или ES
px DW ? ; Параметр 1 ; вызывающей и вызываемой стороны и пишем и
py DW ? ; Параметр 2 ; читаем параметры
re DW ? ; Результат
PARSEG ENDS
```

31) Способы сохранения и восстановления состояния вызывающей программы (кто выполняет и в чьей памяти)

Запоминание и восстановление состояния вызывающей программной единицы (т.е. регистров).

Вариант запоминания регистров определяется следующим

- кто (вызывающая / вызываемая программная единица) является владельцем области сохранения
- кто из них фактически выполняет сохранение регистров

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.

32) Соглашения о связях в Turbo Pascal, Turbo C, Delphi, VS C++

Соглашения о связях в Turbo Pascal

Передача параметров через стек в порядке загрузки слева направо. Возврат результата:

- скалярные результаты по 2, 1 байт через AX (AL), указатели (4 байта) через DX:AX (строка указатель) - Real (6 байтовых паскальские) в DX:BX:AX (это не адресация, просто возврат через эти регистры делится на куски по два байта в таком порядке, сверху такой же смысл)

Сохранение результатов (S I, D I, B P) в подпрограмме.

За уничтожение параметров в стеке отвечает вызываемая программа, например используя RET N (N — количество байт, которые надо очистить)

Соглашения о связях в Turbo C

Передача параметров через общий стек. Помещаются параметры из списка параметров функции в стек в порядке справа налево. Если результат возвращается через имя функции, то он помещается, как правило, в регистр AX или DX:AX.

Сохранение регистров внутри подпрограммы (BP, SP, CS, DS, SS, SI, DI), т.е. их

поместить в стек в начале подпрограммы, если их значения могут изменяться в подпрограмме. За уничтожение параметров в стеке отвечает вызывающая сторона. Освобождение стека выполняет вызывающая сторона командой `ADD SP, N` (`N` — количество байт, которые надо очистить).

33) Команды сдвига.

`SHL операнд, количество_сдвигов`
`SHR операнд, количество_сдвигов`

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. Указанное действие повторяется количество раз, равное значению второго операнда.

`SAL операнд, количество_сдвигов`
`SAR операнд, количество_сдвигов`

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

`rol операнд, количество_сдвигов`
`ror операнд, количество_сдвигов`
`rcl операнд, количество_сдвигов`
`rcr операнд, количество_сдвигов`

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`. Указанные действия повторяются количество раз, равное значению второго операнда.

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. Указанные действия повторяются количество раз, равное значению второго операнда.

34) Команды логических операций.(из евмхыча добавить)

AND	Выполняет операцию логического И между двумя операндами(результат 1 операнд)
OR	Выполняет операцию логического ИЛИ между двумя операндами(результат 1 операнд)
XOR	Выполняет операцию исключающего ИЛИ между двумя операндами(результат 1 операнд)

NOT	Выполняет операцию логического отрицание (НЕ) единственного операнда(результат 1 операнд)
TEST	Выполняет операцию логического И между двумя операндами, устанавливает соответствующие флаги состояния процессора, но результат операции не записывается вместо операнда получателя данных
BT, BTC, BTR, BTS	Копирует бит операнда получателя, номер n которого задан в исходном операнде, во флаг переноса (CF), а затем, в зависимости от команды, тестирует, инвертирует, сбрасывает или устанавливает этот же бит операнда получателя

35) Команды обработки строк и префиксы повторения.

Все команды имеют по 3 разновидности:

- 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.
- В командах операнд Источн, если он есть, задается DS:SI, а операнд Приемн, если он есть, задается ES:DI.
- После выполнения команды автоматически меняются значения SI, если есть Источн, и DI, если есть Приемн, для подготовки к обработке очередных элементов строк (см. описания команд).
- Перед командами обработки строк можно записывать префиксы повторения REP | REPE или REPZ | REPNE или REPNZ.

ПРЕФИКСЫ ПОВТОРЕНИЯ

- REP <команда> - выполнять команду CX раз.
- REPE <команда> - выполнять команду пока CX<>0 перед и (REPZ <команда>) FZ=1 после выполнения команды.
- REPNE <команда> - выполнять команду пока CX<>0 перед и (REPZ <команда>) FZ=0 после выполнения команды.

36) Команды пересылки строк.

Команды пересылки строк MOVSB, MOVSW, MOVS, MOVSD.

- 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.

```
std                ; df := 1
add    edi, есх ; приемник
dec    edi
add    esi, есх ; источник
dec    esi
        ; df - справа налево
norev:
```

rep movsb; из esi в edi записывает байты esx - сколько
последовательно если df= 0 слева направо если df = 1 справа налево

37) 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.

Функция:

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

2.2. Сравнение слов CMPSW

Функция:

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

2.3. Сравнение байтов или слов CMPS Приемн,Источн

Функция:

- а) Та же, что у CMPSB, если тип Источн и Приемн = BYTE
а) 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.

Пример. Сравнить строки Str1 и Str2 из 8 символов и
установить: N=-1 при Str1<Str2,
N=0 при Str1=Str2,
N=1 при Str1>Str2. *

```
DSEG SEGMENT PARA PUBLIC 'DATA'
Str1 DB '....._i_'
N DB ?
DSEG ENDS
```

```
ESEG SEGMENT PARA PUBLIC 'DATA'
Str2 DB '....._I_'
ESEG ENDS
```

```
M2: MOV SI,OFFSET Str1
MOV DI,OFFSET Str2
CLD
MOV CX,8
MOV N,-1
REPE CMPS Str1,Str2 ;Str2 - Str1
JB M202 ;Str2 < Str1
```

```

        JE M201          ;Str2 = Str1
        INC N
M201:   INC N
M202:   JMP $+2

```

38) Команды сканирования строк.

Команды сканирования строк SCASB, SCASW, SCAS, SCASD.
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.

Сканирования байтов

SCASB

Функция:

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

3.2. Сканирования слов

SCASW

Функция:

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

3.3. Сканирования байтов или слов

SCAS Приемн

Функция:

- а) Та же, что у SCASB, если тип Приемн = BYTE
а) Та же, что у SCASW, если тип Приемн = WORD

Пример. Найти длину строки Str3, заканчивающейся кодом 0,
и число пробелов в конце строки

*

```

DSEG SEGMENT PARA PUBLIC 'DATA'
SX      DB      'SS      ',0
LSX     DW      0
XSX     DW      0
DSEG ENDS

```

```

M3:     MOV DI,OFFSET SX
        PUSH ES
        PUSH DS
        POP ES
        ASSUME ES:DSEG ; ! ДИРЕКТИВА НЕОБХОДИМА
        MOV AL,0
        CLD
        MOV CX,-1 ; если задать CX:=0, то
        REPNE SCAS SX ; эта команда будет пропущена
        SUB DI,2
        NEG CX

```

```

SUB CX,2
MOV LSX,CX
STD
MOV AL,' '
REPE SCASB
INC CX
NEG CX
ADD CX,LSX
MOV XSX,CX
JMP M5

```

39) Команды загрузки строк.

Команды загрузки строк LODSB, LODSW, LODS, LODSD.
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.

Загрузки байтов

LODSB

Функция:

- | | |
|------------------------|-------------------------|
| а) При DF=0 (исп. CLD) | б) При DF=1 (исп. STD) |
| - AL:=Байт[DS:SI] | - AL:=Байт[DS:SI] |
| - SI:=SI+1 | - SI:=SI-1 |

4.2. Загрузки слов

LODSW

Функция:

- | | |
|------------------------|-------------------------|
| а) При DF=0 (исп. CLD) | б) При DF=1 (исп. STD) |
| - AX:=Слово[DS:SI] | - AX:=Слово[DS:SI] |
| - SI:=SI+2 | - SI:=SI-2 |

4.3. Загрузка байтов или слов

SCAS Источн

Функция:

- | |
|---|
| а) Та же, что у LODSB, если тип Источн = BYTE |
| а) Та же, что у LODSW, если тип Источн = WORD |

5. Команды сохранения строк STOLS, STOSB, STOSW.

5.1. Сохранения байтов

STOSB

Функция:

- | | |
|------------------------|-------------------------|
| а) При DF=0 (исп. CLD) | б) При DF=1 (исп. STD) |
| - Байт[ES:DI]:=AL | - Байт[ES:DI]:=AL |
| - DI:=DI+1 | - DI:=DI-1 |

5.2. Сохранения слов

STOSW

Функция:

- | | |
|------------------------|-------------------------|
| а) При DF=0 (исп. CLD) | б) При DF=1 (исп. STD) |
| - AX:=Слово[ES:DI] | - AX:=Слово[ES:DI] |
| - DI:=DI+2 | - DI:=DI-2 |

5.3. Сохранения байтов или слов

STOS Приемн

Функция:

- а) Та же, что у STOSB, если тип Приемн = BYTE
а) Та же, что у STOSW, если тип Приемн = WORD

```

Пример. Закодировать строку      SourceText  DB 'ABACADAA',
используя таблицу кодирования CodeTable DB 'BDAC',
в строку      DestText    DB 'BDBABCBB' *

```

```

TSV          STRUC
VectorSourceText  DD ?
VectorDestText    DD ?
VectorCodeTable   DD ?
TSV          ENDS

```

```
CodeTable DB 'BDAC' ; в сегменте CODE
```

```
ESEG SEGMENT PARA PUBLIC 'DATA'
DestText      DB  '*****' ; в сегменте ESEG
ESEG ENDS
DSEG SEGMENT PARA PUBLIC 'DATA'
SourceText    DB  'ABACADAA' ; в сегменте DSEG
```

```
SV TSV <SourceText, DestText, CodeTable>
DSEG ENDS
```

```

.286
M5:      PUSH 8          ; директива .286 необходима, т.к. 8 и
        PUSH OFFSET SV ; OFFSET SV - непосредственные операнды
        CLD
        CALL CODER
        SUB  SP,4

        MOV  AH,4CH
        INT  21H
BEGIN   ENDP

```

```

CODER      PROC NEAR          ;Установить наблюдение:
LStr       EQU [BP+6]
AddrVectors EQU [BP+4]
            PUSH BP           ;>WB SOURCETEXT L8
            MOV BP,SP         ;>WB DESTTEXT L8
            PUSH ES           ;>WB CODETABLE L4
            PUSH DS           ;>WW SS:SP L8
            MOV CX,LStr       ;[BP+6]
            MOV BX,AddrVectors ;[BP+4]
            LDS SI,[BX].VectorSourceText
            MOV BP,DS
            LES DI,[BX].VectorDestText
            LDS DX,[BX].VectorCodeTable
            MOV BX,DX
            MOV DX,DS
            MOV DS,BP
MCODER:    LODSB              ;DSEG
            SUB AL,'A'
            MOV DS,DX
            XLAT               ;CODE
            MOV DS,BP
            STOSB              ;ESEG
            LOOP MCODER
            POP DS
            POP ES
            POP BP
            RET

```

```
CODER    ENDP  
  
CODE     ENDS  
        END    BEGIN
```

40) 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.

41) 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.

42) 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)

43) Описания макроопределений (макрофункций и макропроцедур) и макрокоманд.

Макроопределения — специальным образом оформленная последовательность предложений языка ASM. Код управления которой ASM (точнее его часть порождает — макрогенератор) макрорасширение макрокоманд.

Макрорасширения (результат макроподстановки)— последовательность предложений языка, порождающаяся макрогенератором при обработке макрокоманды. /* под управлением макроопределения.*/

Макрокоманда (ссылка на макрос) — предложение в исходном тексте программы, которое воспринимается макрорасширением, как приказ построить макрорасширение и вставить на её место (“вызов” макроса).

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

;Формат макрокоманды:
<имя макроса> <фактические параметры через запятую и/или пробел>
;Описание макроса:
PRINTSTR MACRO STR
MOV DX,STR
MOV AH,9
INT 21H
ENDM
;Вывод строки по смещению R.
;Пример вызова:
LEA BX, SOME_STR
PRINTSTR BX ; макрокоманда
```

Замечания по исполнению макроопределений: 1. длина формальных параметров ограничена длиной строки
2. нет ограничений, кроме физических
3. макроопределения рекомендуется размещать в начале программы
4. макрос можно определять внутри другого макроса, но тогда вложенный макрос будет доступен только после первого вызова внешнего макроса
5. определение макроса с именем, которое ранее уже было использовано (переопределение макроса) затирает предыдущее определение
6. макрос можно уничтожить с помощью PURGE <имя_макроса>{, <имя макроса>}
7. макроопределения можно хранить в отдельном txt-файле и включать в программу с помощью INCLUDE

44) Директива INCLUDE.

```
INCLUDE <имя файла без кавычек>
```

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

45) 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. Например, RECUR зразвернется в

```

MOV AX, 3
MOV AX, 2
...

```

46) Параметры в макросах.

```

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

```

Пример вызова макроса с фактическими параметрами 1, 2: MACRONAME 1,2
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.

```

;Пример макроса с параметром:
PRINT_STR MACRO STR ; STR параметр
    MOV AH, 9
    MOV DX, STR
    INT 21h
ENDM

;Пример макроса без параметров:
EXIT_APP MACRO
    MOV AH,4C
    INT 21h
ENDM

```

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.

47) Директива LOCAL.

```

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))

48) Директивы условного ассемблирования IF, IFE, IF2, IFIDN/IFIDNI, IFDIF /IFDIFI, IFDEF, IFNDEF 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:

```
IFDEF A
    ADD AX, A
ELSE
    EXITM
ENDIF
```

В примере к AX будет прибавлено A только если имя A определено, иначе совершится выход из макроса.

49) 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.

IFNB arg

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

```
SHOW MACRO REG
    IFB REG
        DISPLAY 'НЕ ЗАДАН РЕГИСТР'
        EXITM ENDF
    ...
ENDM
```

50) Директивы IFIDN и IFDIF в макроопределениях.

IFIDN, — истинно, если строки (любой текст) S1 и S2 совпадают. (ifIDeNtical)

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

<, > не метасимволы, они обязательны.

```
SOMETHING MACRO
  IFIDN <A>, <B>
    DISPLAY 'ЭТО НИКОГДА НЕ ВЫПОЛНИТСЯ'
    EXITM
  ENDF
...
ENDM
```

51) Операции ;; % & < > ! в макроопределениях.

1. **;;** — комментарий.
формат: ;; text
2. **%** — если требуется вычисление в строке некоторого константного выражения или передача его по значению в макрос.
формат: %ВЫРАЖЕНИЕ, например: K EQU 5, %K+2 ; => 7
3. **!** — 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!&”
4. **&** — 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. Алсо, это интерполяция строк.
5. **< >** — содержит часть текста программы (в макрорасширении заменяется ровно на то что в скобках). Внутри скобок последовательность любых символов.
формат: <text>, например <5+2>

```
DEBUGMSG MACRO WHERE, WHAT
  DBGMSG&WHERE DB "&WHAT& occured at &WHERE&$"
ENDM
; Вызов DEBUGMSG 135,<A huge error> развернется в DBGMSG135 DB "A huge error
occured at 135$"
```

52) Блок повторения REPT.

```
REPT WWW
; ...тело блока
ENDM
```

WWW — выражение, значение которого (ЦБЗ) определяет число копирований макроопределения в макрорасширение.

53) Блок повторения IRP/FOR.

```
IRP FORM, <FACT1, FACT2, ... > (угловые скобки обязательны)
; ...тело блока
ENDM
```

FORM — формальный параметр, который используется в теле блока: FACT1, ..., FACTi — подставляемый на место формального при i-ом копировании.

```
IRP X, <0,1,2>
    A&X DB X
ENDM
```

;После развертки (макрорасширение):

```
A0 DB 0
A1 DB 1
A2 DB 2 ...
```

54) Блок повторения IRPC/FORC.

```
IRPC param, str
; ...тело блока
ENDM
```

param — формальный параметр, который используется в теле блока, str — набор символов, i-ый символ которой при i=1, 2,...,<длина_этого_набора> подставляется на место формального параметра при i-ом копировании.

```
IRPC X, ABC
    X DB 'X'
ENDM
```

;После развертки (макрорасширение):

```
A DB 'A'
B DB 'B'
C DB 'C' ...
```

55) Блок повторения WHILE.

```
WHILE выражение
блок
ENDM
```

Блок повторяется, пока выражение не станет равным нулю.

56) Директива EQU в MASM.

Директива EQU присваивает метке значение, которое определяется как результат целочисленного выражения в правой части. Результатом этого выражения может быть целое число, адрес или любая строка символов:

метка equ выражение

```
truth equ 1
```

```

message1 equ    'Try again$'

var2    equ     4[si]

        cmp     ax,truth    ; cmp ax,1

        db      message1    ; db 'Try again$'

        mov     ax,var2     ; mov ax, 4[si]

```

Директива EQU чаще всего используется с целью введения параметров, общих для всей программы, аналогично команде #define препроцессора языка C.

57) Директива TEXTEQU в MASM32.

Существует три формата директивы textequ:

```

имя      TEXTEQU    <текст>
имя      TEXTEQU    текстовый__макрос
имя      TEXTEQU    %константное_выражение

```

В первом случае символу присваивается указанная в угловых скобках <. . . > текстовая строка. Во втором случае — значение заранее определенного текстового макроса. В третьем случае — символической константе присваивается значение целочисленного выражения.

В приведенном ниже примере переменной prompt1 присваивается значение текстового макроса continueMsg:

```

continueMsg    TEXTEQU    <"Хотите продолжить (Y/N)?">
. data
prompt1        BYTE continueMsg

```

Символ, определенный с помощью директивы TEXTEQU, можно переопределить в программе в любой момент.

58) Директива = в MASM.

Директива = эквивалентна EQU, но определяемая ею метка может принимать только целочисленные значения. Кроме того, метка, указанная этой директивой, может быть переопределена.

Каждый ассемблер предлагает целый набор специальных предопределенных меток — это может быть текущая дата (@date или ??date), тип процессора (@cpu) или имя того или иного сегмента программы, но единственная предопределенная метка, поддерживаемая всеми рассматриваемыми нами ассемблерами, — \$. Она всегда соответствует текущему адресу. Например, команда

```

jmp $

```

выполняет безусловный переход на самую себя, так что создается вечный цикл из одной команды.

59) Типы макроданных text и number (см листинг) !!!---

60) Именованные макроконстанты MASM32 !!!

61) Макроимена, числовые и текстовые макроконстанты - значения. !!!

62) Директивы echo и %echo

Директива echo выводит следующую после неё и пробела часть строки на экран

Echo privet ya dibil

%echo вычисляет свой параметр, т.е. значение переменной T, и выводит его

%echo @CatStr(A,+,B) OUT: A+B %echo вычисляет свой параметр, т.е. вызывает макрофункцию @CatStr, которая подставляет вместо себя результат своей работы, т.е. строку A+B, а %echo выводит эту строку.

63) Способы вывода значений макропеременных и макроконстант с пояснениями !!!

display

64) Операций в выражениях MASM: !!!

- Арифметические операции.

Общие арифметические операции используются для работы с константами, значениями идентификаторов и значениями других общих арифметических операций. Общими операциями являются операции сложения, вычитания, умножения и деления.

+ - * /

- Логические операции.

Логические арифметические операции позволяют вам выполнять операции булевой алгебры. Каждая из этих операций выполняется поразрядно, то есть, логическая операция выполняется по одному биту. Логические операции приведены в следующей таблице.

Логические арифметические операции Таблица 5.16

Т	
Выражение	Значение
NOT выражение	Поразрядное дополнение выражения.
выражение_1 AND выражение_2	Поразрядная операция "И".

выражение_1 OR выражение_2	Поразрядная операция "ИЛИ".
выражение_1 XOR выражение_2	Поразрядная операция "исключающее ИЛИ".

- Операции отношений.

Операции сравнения позволяют сравнить два выражения и проверить их равенство или неравенство или что одно из них больше или меньше другого. Эти операции равны -1, если условие истинно (True), или 0 в противном случае. Следующая таблица показывает, как можно использовать эти операции.

Операции сравнения

Таблица 5.18

Выражение	Значение
выражение_1 EQ выражение_2	-1, если выражение_1 равно выражению_2, в противном случае 0.
выражение_1 NE выражение_2	-1, если выражение_1 не равно выражению_2, в противном случае 0.
выражение_1 GT выражение_2	-1, если выражение_1 больше выражения_2, в противном случае 0.
выражение_1 GE выражение_2	-1, если выражение_1 больше или равно выражению_2, в противном случае 0.
выражение_1 LT выражение_2	-1, если выражение_1 меньше выражения_2, в противном случае 0.
выражение_1 LE выражение_2	-1, если выражение_1 меньше или равно выражения_2, в противном случае 0.

Операции EQ или NE интерпретируют выражения, как числа без знака. Например, -1 EQ 0ffffh имеет значение -1 (если только вы не выбрали процессор 80386 или не используете режим Ideal; в последнем случае значение -1 имеет выражение -1 EQ 0ffffffffh).

Операции Gt, GE, LT и LE интерпретируют выражения, как числа со знаком. Например, 1 GE -1 имеет значение -1, но 1 GE 0ffffh имеет значение 0.

- Операции, возвращающие значения

!!!

- Операции присваивания атрибута.

65) Подготовка ассемблерных объектных модулей средствами командной строки для использования в Delphi и VS C++.

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

Команда получения объектного модуля для последующего использования

Если их несколько можно написать батник

66) Добавление ассемблерных модулей в проект консольного приложения VS C++

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

Пример

Extern void ppstart(), "C"

67) Добавление ассемблерных модулей в проект консольного приложения Delphi

{ \$L ppstart.obj }

procedure start; external;

для вызова таким образом должен быть прописан объектный модуль из которого нужно вытащить нужную нам функцию

68) Использование ассемблерных вставок в модулях .cpp. !!!

Что за х

69) Вызов из ассемблерной подпрограммы C в VS C++. !!!

Что за х

70) Передача глобальных данных, определённых в консольной программе VS C++, в ассемблерный модуль. !!!

Что за х

71)Передача глобальных данных, определённых в ассемблерном модуле в консольный модуль .cpp VS C++. !!!

Что за х

72)Средства отладки в CodeView. Примеры. !!!

Что за х

73)Средства отладки в VS C++. Примеры. !!!

Что за х

74)Получение дизассемблированного кода в VS C++ !!!

Что за х