Тема 2.

Разработка кода программного продукта на уровне модуля

Занятие 2.1.

Программирование на языке Ассемблера. Структура языка

Цели занятия :

1. Изучить сущность и структуру языка ассемблера, его основные директивы, варианты их применения при разработке и отладке программного обеспечение.

Учебные вопросы:

1. Сущность и структура языка ассемблера

2. Основные директивы языка ассемблера

3. Структура программ микропроцессорных систем на языке ассемблера

1. Сущность и структура языка ассемблера

АССЕМБЛЕР (assembler – сборщик)

это язык *символического* кодирования,
то есть, машинный язык в *символьной* форме, понятной человеку.

Для процессоров *ix86* содержит более *100* базовых символических команд, в соответствии с которыми генерирует более *3800 машинных* команд.

Для записи исходных модулей программ использует *операторы*.

Общие положения о конструкциях ассемблера

Исходный модуль программы

Оператор

Оператор

Оператор

Оператор

.....

Оператор

Исходный модуль - это последовательность ***операторов*** языка. Каждый оператор занимает *одну* строку в тексте программы.

Различают операторы:

- исполняемые;
- неисполняемые

Разновидности операторов ассемблера

Исходн
програ

Исполняемые операторы - транслируются в машинные коды. Они управляют работой процессора, поэтому их еще называют *ком*

Неисполняемые операторы - машинный код не генерируют. Они управляют не процессором, а программой, поэтому их называют *псевдооператорами,* или просто *директивами.*

Оператор
Оператор
Оператор
.....
Оператор

Алфавит языка:

1 - прописные и строчные латинские буквы;

2 - цифры ;

3 - специальные символы:

+ - * / = ( ) [ ] < > $ @ ? & . , ; : ’ ”

$ - текущее значение адреса (счетчик адреса) – специфический вид операнда. Встретив такой символ в программе, транслятор автоматически вместо него подставляет значение IP.

Примеры:

```
JMP $       ; Команда вызывает переход на саму себя,
              т.е. бесконечный цикл.
JMP $ + 3   ; Переход по адресу IP + 3
```

@ - указывает на соответствующие физические адреса:
@code – сегмента кода;
@data – сегмента данных и т.д.
Определяются директивой MODEL

Структура командных операторов (команд):

[Метка:]<Мнемокод>[Операнд][;Коммент]

обязательное

то есть, содержит поля:

поле метки

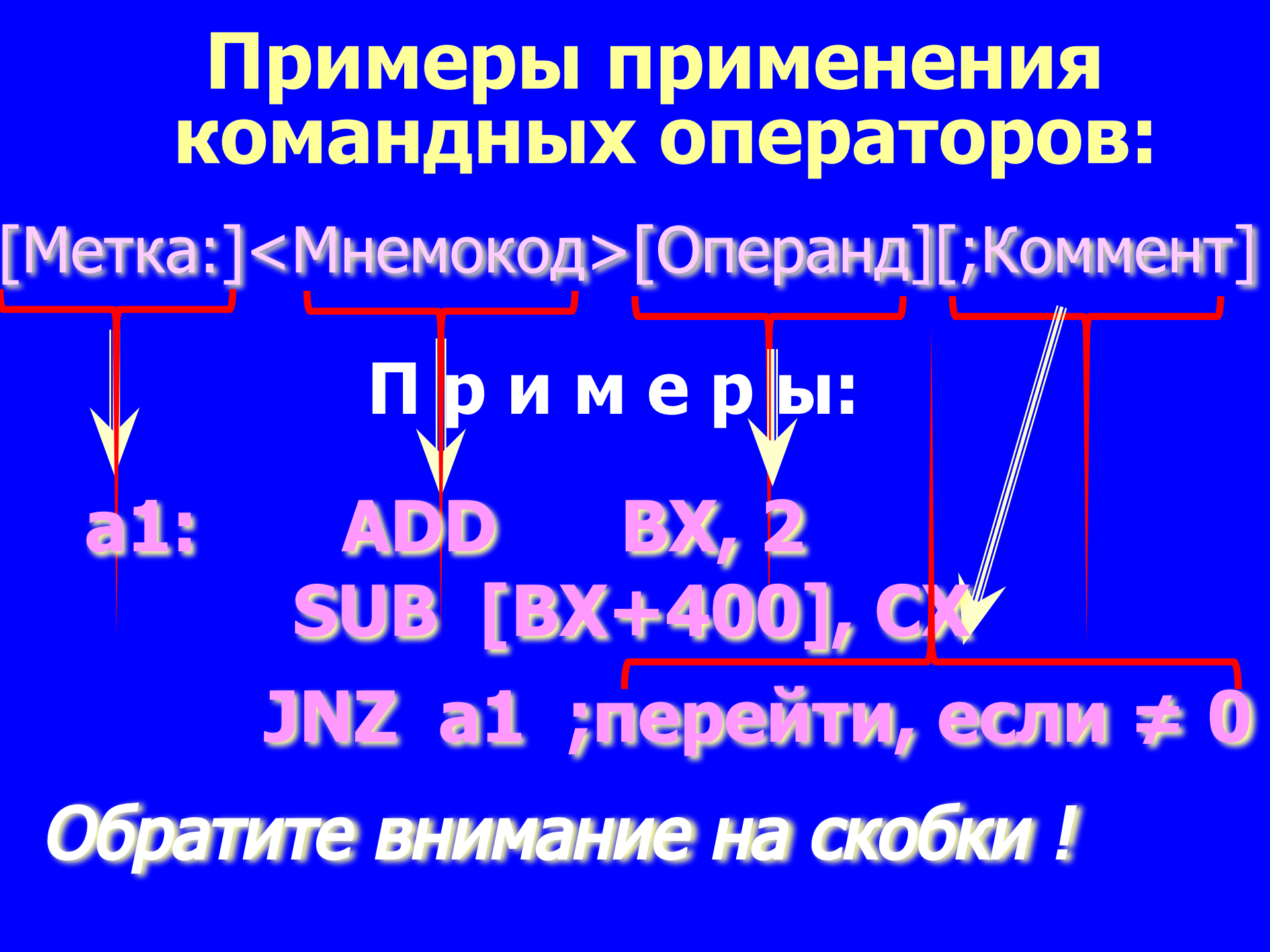
поле мнемокода

поле операнда(ов)

поле комментариев

Примеры применения командных операторов:

[Метка:]<Мнемокод>[Операнд][;Коммент]



Обратите внимание на скобки !

Особенности поля метки

Служит для присвоения имени команде.

Должно заканчивается двоеточием.

Рекомендации по выбору имени :

- Количество символов **≤ 31**;
- Нельзя начинать с цифры (цифры можно только в **Debug**);
- Не использовать имен **РОНов** и **имен команд** (мнемокодов).

Поле комментариев

Должно начинаться точкой с запятой и отделяться хотя бы одним пробелом от предыдущего поля.

Примеры комментариев:

POP AX ;Возврат параметра Pн в регистр AX

;
;

Пустые строки

; Расчет зоны поражения ЗРК

Самостоятельная строка

Псевдооператоры (директивы)

Управляют работой ассемблерной *программы* (но не работой процессора)

При этом позволяют:

- **Определять сегменты и процедуры;**
- **Присваивать имена командам и элементам данных;**
- **Резервировать рабочие области памяти и выполнять много других "хозяйственных" задач;** р

Для микропроцессоров семейства *ix86* предусмотрено более *60* различных директив !

2. Основные директивы языка ассемблера

Наиболее часто используемые псевдооператоры (директивы)

1-я группа

Псевдооператоры данных :

- **ASSUME; SEGMENT;**
- **ENDS; PROC; ENDP;**
- **END; INCLUDE;**
- **DB; DW; DD; DQ**
- **и др.**

2-я группа

Псевдооператоры управления листингом :

- **PAGE;**
- **TITLE;**
- **SUBTTLE**

Назначение директив ассемблера

ASSUME - связывает имя сегмента программы с сегментным регистром процессора;

SEGMENT - определяет границы сегмента программы, т. е. называет *сегментом программы* группу операторов;

PROC - определяет (присваивает) имя процедуре (т. е. подпрограмме);

ENDP - конец подпрограммы;

ENDS - конец сегмента;

END - конец программы.

Назначение директив ассемблера

INCLUDE - при трансляции **подключает** (т. е. вставляет) текст из другого файла в текущий файл программы;

DB; DW; DD; DQ; DT - определяют и резервируют требуемое количество соответственно: *байт*, *слов*, *двойных слов*, *4-кратных слов*, *10 байт* .

Структура директивы SEGMENT

<Имя> SEGMENT [<Тип выравнивания>]..

.....

.....

.....

<Имя> ENDS

Может быть

Выравнивает физический адрес

Означает

Страница	⇒	**PAGE**	**XXX00**
Параграф	⇒	**PARA**	**XXXX0**
Слово	⇒	**WORD**	**XXXXe**
Байт	⇒	**BYTE**	**XXXXX**

т. е. указывает, как начинать сегмент

Структура директив DB, DW, DD

[Имя] **DB** [выражение] [, ...]
[Имя] **DW** [выражение] [, ...]
[Имя] **DD** [выражение] [, ...]

Пример:

```
Tab_1 DB 0,0,0,0,8,-13                ;Таблица байтов
      DB 40,55,-7,63,63,63
Tab_2 DW 1025,507,-2014,809           ;Таблица слов
```

Использование конструкции DUP

```
Tab_1 DB 4 DUP (0),8,-13,40,55, -7, 3 DUP (63)
      DW 32 DUP (?)  ;Зарезервировано 32 слова
```

Особенности применения директив определения сегментов

В ранних версиях ассемблеров –

только стандартные директивы
SEGMENT, ENDS, ASSUME

В новых версиях ассемблеров –

либо стандартные директивы,
либо простейшие, такие как
.MODEL,
.STACK,
.DATA,
.CODE и др.

Разновидности моделей памяти

.MODEL tiny – *минимальная*. Код программы и данные размещаются в одном сегменте размером до 64 Кбайт.

.MODEL small – *малая*. Код и данные размещаются в разных сегментах размером до 64 Кбайт каждый. Наиболее эффективная и чаще всего используется.

.MODEL medium – *средняя*. Код программы и данные размещаются в разных сегментах. Размер сегмента кода > 64 Кбайт, размер сегмента данных = 64 Кбайт.

.MODEL compact – *компактная*. Код программы и данные размещаются в разных сегментах. Размер сегмента кода ≤ 64 Кбайт, размер сегмента данных > 64 Кбайт.

.MODEL large – *большая*. Код программы и данные размещаются в разных сегментах размером > 64 Кбайт каждый.

3. Структура программ МПС на языке ассемблера

Структура модуля программы, разбитого на сегменты

```
Stack  SEGMENT
.....
.....
Stack  ENDS
Data  SEGMENT
.....
.....
Data  ENDS
Code  SEGMENT
.....
.....
Code  ENDS
```

Пример сегмента стека

Пример сегмента данных

Пример сегмента кода

Пример фрагмента программы

```
TITLE  Ex_prog - программа эксперимента
Stack  SEGMENT          ;Начало сегмента Stack
DB       64 DUP (?)     ;Зарезервировано 64 байта
Stack  ENDS             ;Конец сегмента Stack
Data   SEGMENT          ;Начало сегмента Data
Source DB 10,20,30,40   ;Задан массив Source
Dest     DB 4 DUP (?)   ;Зарезервировано 4 байта для
                        ; массива Dest
Data   ENDS             ;Конец сегмента Data
SUBTTLE - Основная программа
Code     SEGMENT ;Начало сегмента кода
        ASSUME CS:Code, DS:Data, SS:Stack
```

Примеры вложенных процедур

```
SegCode  SEGMENT
    zrk_1  PROC
          ..................................
          raketa  PROC
          ..................................
          ..................................
          ..................................
          ..................................
          raketa ENDP
          ..........................
    zrk_1 ENDP
SegCode  ENDS
```

Программа изменения массива 1

```
TITLE  Ex_prog - Обработка массива (таблицы)
Stack  SEGMENT      ;Начало сегмента стека
DB      64 DUP (?)  ;Зарезервировано 64 байта
Stack  ENDS         ;Конец сегмента стека
Dseg   SEGMENT     ;Начало сегмента данных
Source DB 10,20,30,40  ;Задан массив Source
Dest    DB 4 DUP (?)  ;Зарезервировано 4 байта
Dseg    ENDS          ;Конец сегмента данных
SUBTTLE -  Основная программа
Cseg    SEGMENT ;Начало сегмента кода
Our_prog PROC   FAR ;Начало подпрограммы
        ASSUME CS:Cseg, DS:Dseg, SS:Stack
;  Обеспечение передачи управления программе Debug
  PUSH DS  ;В стек номер блока адреса возврата
  MOV AX,0 ;Обнулить регистр
  PUSH AX   ; В стек нулевое смещение адреса возврата
```

продолжение программы


```
MOV AX, Dseg  ; Инициализировать
MOV DS, AX    ;  регистр DS
;Присвоение элементам таблицы Dest нулевых значений
MOV  Dest,  0       ;Первому байту (нулевому элементу)
MOV  Dest+1,  0   ;Второму байту (первому элементу)
MOV  Dest+2,  0   ;Третьему байту (второму элементу)
MOV  Dest+3,  0   ;Четвертому байту (третьему элементу)
;Копировать табл.  Source в табл. Dest   в обратном порядке
MOV  Al,  Source    ;Скопировать 1-й байт
MOV  Dest+3,  Al
MOV  Al,  Source+1 ;Скопировать 2-й байт
MOV  Dest+2,  Al
MOV  Al,  Source+2 ;Скопировать 3-й байт
MOV  Dest+1,  Al
MOV  Al,  Source+3 ;Скопировать 4-й байт
MOV  Dest,  Al
RET   ;Возврат управления отладчику DEBUG
```

окончание программы


```
Our_prog ENDP  ;Конец подпрограммы
Cseg     ENDS  ;Конец сегмента кода
         END Our_prog  ;Конец программы
```

Программа вывода символов на экран

```
masm                ;задание режима работы TASM: ideal или masm
model small         ;задание модели памяти типа small
.stack 256
.data   Stroka     DB  55h, 6Eh, 69h, 76h, 65h, 72h, 73h, 69h, 74h, 65h, 74h
 Long = $-Stroka       ;вычисление размера (длины) массива
.code
 start:  MOV AX, @data         ;загрузка адреса начала сегмента данных в регистр DS
    MOV DS, AX            ;через регистр AX (для вывода массива Stroka на экран )
    MOV CX, Long            ;длина массива заносится в счетчик циклов
    LEA SI, Stroka          ;загрузка адреса начала массива в регистр SI
    MOV AH, 02h       ;загрузка функции DOS 02h для вывода символа на экран
m4:  MOV DL, byte ptr [SI]    ;загрузка выводимого символа в регистр DL
       INT 21h
       INC SI
       LOOP m4
exit:
;выход из программы
MOV AX, 4C00h ;установка функций 4Ch и 00h в регистрах AH и AL
INT 21h
end  start          ;конец программы с точкой входа start
```