2. Структура программ на языке ассемблера

Ассемблерная программа представляет собой текст, **строго** разбитый на строки и оформленный по специальным правилам. Каждая строка содержит либо ассемблерную инструкцию, либо управляющую директиву языка. Этот входной текст обрабатывается программой-ассемблером с переводом текста в двоичное представление.

Весь текст обычно разбивается на две основные части:

- описание данных, используемых в программе (набор управляющих директив объявления данных); такой фрагмент часто называют термином «сегмент данных»
- набор ассемблерных инструкций, определяющих операции с данными в соответствии с реализуемым алгоритмом – сегмент команд или кодовый сегмент

Ассемблерная инструкция (команда) – это символьная запись соответствующей машинной команды. Каждая строка с ассемблерной инструкцией может содержать до четырех компонентов, порядок следования которых **строго определен**:

- на первом месте записывается так называемая **метка** команды (она может отсутствовать); метка заканчивается двоеточием
- после метки через разделитель (хотя бы один пробел) записывается **мнемонический код команды** (МКК)
- после МКК через пробел задаются **операнды** команды (они могут отсутствовать); если операндов два, то они отделяются друг от друга запятой
- в конце строки с помощью точки с запятой можно обозначить необязательное поле **комментария**

Тем самым, **общий формат** ассемблерной команды выглядит следующим образом:

метка: МКК операнд1,операнд2 ; комментарий

Каждое поле имеет свою смысловую нагрузку:

а) Метка – короткое символьное имя, которое заменяет адрес, назначаемый данной командой. Метка используется в операндах команд для ссылки на помеченную команду. Это позволяет в ассемблерных программах не работать непосредственно с адресами памяти. Замена меток адресами выполняется при ассемблировании. Метка представляет собой набор латинских букв и цифр, как привило, начинающийся с буквы. Регистр безразличен. **Важно**: все метки в программе должны быть разными!

б) Мнемонический код команды заменяет двоичный код более понятным символьным обозначением; набор МКК зафиксирован в языке аналогично служебным словам языков высокого уровня. При ассемблировании МКК заменяется двоичным эквивалентом

в) Операнды задают объекты команды и могут включать в себя:

- непосредственные константы разных типов
- имена регистров (например, AX, ECX, BL)
- метки команд
- символьные имена переменных (они заменяют адреса размещения в памяти обрабатываемых данных)

В качестве операндов можно использовать следующие типы констант:

- числовые двоичные константы как набор 0 и 1, заканчивающийся символом b (от binary, двоичный), например: 00101011b
- десятичные числа в обычном представлении
- шестнадцатеричные константы вида 2A1Ch с обязательным оконечным символом h (от hexadecimal), причем, если число начинается с A-F, то спереди обязательно ставится незначащий ноль: 0B2h – число, а B2h – символьное имя
- Символьные константы с помощью одинарных или двойных кавычек могут задавать один или несколько символов, например ‘A’, “Hello”.

Примеры ассемблерных команд:

```
Met1:  ADD   AX, 1   ;комментарий
```

```
MOV   BX, MyName   ; здесь  MyName – имя переменной
```

Директивы (псевдокоманды) используются для управления процессом ассемблирования. В отличие от ассемблерных команд директивам **НЕ соответствуют** никакие машинные команды, поэтому после ассемблирования исходного текста явных следов использованных в нем директив заметно не будет (хотя неявное воздействие на создаваемый код безусловно имеется).

Общий формат директив похож на формат команды:

имя название операнд(ы) ;комментарий

Здесь обязательным полем является только название директивы. Все названия являются служебными словами языка и должны записываться строго по правилам. Основные директивы вводятся в дальнейших разделах пособия по мере необходимости, но уже сейчас надо определить три важнейшие директивы, которые используются для описания данных в программе, то есть переменных и констант. Это следующие директивы:

- объявление байтовых данных – DB (т.е. Define Byte)
- объявление данных размером в слово – DW (т.е. Define Word)
- объявление данных размером в двойное слово – DD (т.е. Define Double)

С помощью одной директивы можно задать как одиночный элемент данных, так и целый набор **<u>однотипных</u>** данных. Можно не определять начальные значения описываемых данных, а только лишь зарезервировать память под последующее использование.

Примеры использования директивы DB:

```
Zero  DB  0    ; определяет область памяти размером в 1 байт с
                 ; начальным значением 0 и именем Zero
Plus   DB  '+'  ; в области памяти с именем Plus записывается код
                  ; символа
Mass   DB   1, 2, 3, 4, 5   ; пять байтов с целыми числами, где имя Mass
                              ; определяет адрес размещения первого числа
  X1   DB   ?     ; только резервирование одного байта без инициализации
```

```
X2  DB  1, ?, ?, ?, 4   ; резервируются 5 байтов, три из которых содержат
                        ; старую информацию
Str  DB  ‘Привет’   ; 6 байтов с начальным адресом Str и кодами
                        ; символов.
```

Для задания набора данных с одинаковыми значениями можно использовать служебное слово DUP (от duplicate, дублировать, повторять):

```
MasZero   DB   10   DUP (0)     ; 10 байтов с нулями
MasX      DB   20   DUP (‘X’)    ; 20 байтов с кодами символа ‘X’
Nabor     DB    5    DUP (1, ‘a’, 2)    ; 5 раз повторяется тройка вида
                                  ; число 1 – код символа ‘a’ – число 2
```

Наконец, можно выполнить объявление данных аналогично заданию двухмерного масива:

```
Mas2D    DB   10   DUP (5  DUP(?))   ; резервировать 10 раз по 5 байтов
```

Использование директив DW и DD практически такое же, только распределение памяти выполняется не по байтам, а по словам или по двойным словам:

```
A   DW   ?   ; резервирование одного слова
B   DW   1234h   ; слово с начальным значением 1234_16, причем
                 ; “переворот” числа выполнит ассемблирующая программа
C   DW   10   DUP (?)      ; резервируются 10 слов (20 байт)
D   DD    5    DUP (?)      ; резервируется 5 двойных слов (20 байт)
```

Константы определяются с помощью директивы EQU:

```
имя     EQU   операнд
M       EQU   100                ; константа M=100
N       EQU   2*M*M - 5*M+1 ; константа N=2*100^2-5*100+1=19 501
STAR  EQU   ‘*’                  ; константа STAR = ‘*’
ERR    EQU   ‘Это ошибка’    ; константа ERR заменяет текст
```