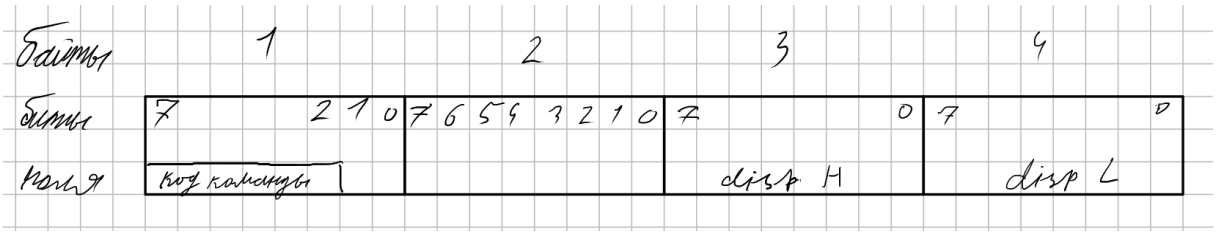
Машинный формат двухадресной команды, для которой один операнд находится всегда в регистре, а второй — в регистре или памяти можно представить следующим обрезом:



...

В ассемблере, как и в языках высокого уровня, могут использоваться именованные константы.

25 — неименованная константа, которой можно присвоить символическое имя.

```
mov ax, 25
const equ 34h ; именованная константа const
mov ax, const
```

Прямая адресация

Если нам известен физический адрес байта, начиная с которого расположен операнд, мы можем его написать прямо в команде:

```
mov ax, es : 0001
```

Можно использовать в качестве прямой адресации символическое имя, которому предварительно поставлен в соответствие некоторый адрес. Это может быть сделано с помощью директив определения данных и памяти, среди которых: `db`, `dw`, `dd` — определить байт, слово и двойное слово.

Если в сегменте `ES` содержится директива `Var_p DW ?`, тогда по команде

```
mov ax, es : Var_p ; ((ES) + Var_p) -> ax
```

Например, если команда имеет ид:

```
mov ax, Var_p; ((DS) + Var_p) -> ax
```

Косвенно-регистровая адресация

Косвенно-регистровая отличается от регистровой тем, что в регистре содержится не операнд, а адрес области памяти, в которой содержится операнд.

```
mov ax, [st]
```

Могут использоваться регистры: si, di, bx, bp, eax, ebx, ecx, edx, ebp, esi, edl.

Не могут: ax, cx, dx, sp, esp.

Адресация по базе со смещением

```
mov ax, [bx + 2] ; ((ds) + (bx) + 2) -> ax
mov ax, [bx] + 2 ; то же самое
mov ax, 2[bx]    ; то же самое
mov ax, [bp + 4] ; ((ss) + (bp) + 4) -> ax
```

6. Прямая с индексированием

```
mov ax, mas[si] ; ((ds) + (si) + mas) -> ax
```

Используется для работы с одномерными массивами и полями структур.

По базе с индексированием

```
mov ax, arr[bx][di] ; ((ds) + (bx) + (di) + arr)
```

Эта адресация позволяет работать с двумерными массивами и со структурами.

особенности использования команд пересылки

1. Нельзя пересылать информацию из одной области памяти в другую. Для этого есть команды работы со строками
2. Нельзя пересылать информацию из одного сегментного регистра в другой. Если очень хочется, то есть регистры общего назначения и стек:

```
push ds
pop es
```

3. Нельзя пересылать непосредственный операнд в сегментный регистр, но если такая необходимость возникает, то нужно использовать в качестве промежуточного один из регистров общего назначения:

```
mov dx, 100h
mov ds, dx
```

4. Нельзя изменять командой mov содержимое регистра cs
5. Размер передаваемых данных определяется типом операндов в команде:

```
x db ? ; x --- адрес одного байта в памяти
y dw ? ; y определяет поле в 2 байта в памяти
mov x, 0 ; очищение одного байта в памяти
mov y, 0 ; очишение двух байтов в памяти
mov ax, 0 ; очищение двухбайтового регистра
mov [si], 0 ; сообщение об ошибке: нам неизвестны размеры полей
```

Для выхода из неопределённости размера полей можно использовать специальный оператор

```
тип ptr выражение
```

где тип — byte, word, dword и т д., а выражение — константа или адрес

```
byte ptr 0 ; 0 применяется как байт
word ptr 0 ; 0 применяется как слово
byte ptr op ; один байт в памяти
mov byte ptr [si], 0 ; = mov [si], byte ptr 0
mov [si] word ptr 0 ; 0 -> ((ds) + (si))
```

7. Если тип обоих операндов определён, то они должны соответствовать друг другу

```
mov ah, 500 ; сообщение об ошибке
mov ax, x ; ошибка: x --- 1 байт, ax --- 2 байта
mov al, r ; ошибка
mov al, byte ptr r ; (AL) = 34h
mov al, byte ptr r+1 ; (AL) = 12h
```

К командам пересылки относят команду обена значений операндов:

```
xchg op1, op2 ; r <-> r or r <-> m
mov ax, 10h
mov bx, 20h
xchg ax, bx ; (ax) = 20h, (bx) = 10h
```

Для перестановки значений байтов внутри регистра используют bswop:

```
(eax) = 12345678h
bswop eax ; (eax) = 78563412h
```

Команды конвертирования:

```
cbw ; безадресная команда, (AL) -> AX
cwd ; (ax) -> dx:ax
cwe ; (ax) -> eax (i386+)
cdf ; (eax) -> edx:eax (i386+)
```

Команды условной пересылки cmovxx:

```
cmovl al, bl ; (al) < (bl) => (bl) -> (al)
```

Загрузка адреса

```
lea op1, op2 ; вычисляет адрес op2 и пересылает первому операнду, который может быть только регистром
```

```
lea bx, m[bx][di]
```

Этапы обработки

Три этапа обработки:

1. Из исходного кода получаем программный код машины. Исходный код может состоять из нескольких модулей
2. Исходные модули объединяются в исполняемый модуль (.exe).

Чтобы выполнить com-файл, нужно выполнить ещё один этап обработки:

3. С помощью системной программы exe2com или в среде разработке с помощью специального ключа

Команды и директивы в ассемблере

Команда на ассемблере состоит из четырёх полей:

[<имя>[:]] <код операции> [<операнды>][; комментарий]

Поля отделяют друг от друга как минимум одним пробелом. В квадратных скобках указаны необязательные поля. Кроме кода операции могут участвовать имя — символическое имя Ассемблера. Имя используется в качестве метки для обращения к этой команде, передачи управления на другую команду, [:] после имени означает, что метка является внутренней. Код операции определяет, какое действие должен выполнить процессор. Поле <операнды> содержит адреса данных, или данные, участвующие в операции, а также место расположения результатов операции. Операндов может быть от 1до 3, они отделяются друг от друга запятой.

Комментарии отделяются кроме пробела ещё “;” и могут занимать всю строку или часть строки. например:

```
jmp m1
```

; команда безусловной передачи управления на команду с меткой.

```
m1: mov ax, bx ; пересылка содержимого регистра bx в регистр ax
```

В комментарии выше будем записывать в виде (BX) AX

Директива

Директива, как и команда, может иметь 4 поля:

```
[<имя>] <код псевдооперации> <операнды> [; комментарии]
```

Здесь

- имя — символическое имя ассемблера
- код псевдооперации определяет назначение директивы

Например:

```
m1 db 1, 0, 1, 0, 1 ; db определяет 5 байтов памяти, заполняет их 0 или 1
соответственно, адрес первого байта --- M1
m2 db ?, ?, ? ; директива db определяет 3 байта памяти без инициализации
```

```
proc ; директива начала процедуры
edp ; директива конца процедуры
```

Исходный модуль на ассемблере — последовательность строк, команд, директив и комментариев.

Исходный модуль просматривается ассемблером, пока не встретится директива end. Обычно программа на ассемблере состоит из трёх сегментов: сегмент стека, сегмент данных, сегмент кода.

```
; сегмент стека
sseg Segment
sseg ends
; сегмент данных
dseg segment
dseg ends
; сегмент кода
cseg segment
cseg ends
end start
```

Назначение сегментов

```
assume ss:sseg, ds:dseg, cs:cseg, es:dseg
```

ассмблера не будет даня ебанулся