**ПЛАН ЗАНЯТИЯ**

**Дисциплина:** МДК.01.04 Системное программирование

**Преподаватель:** Галузин А.Б.

**Курс:** 4

**Группа:** П-40

**Специальность:** 09.02.07 Информационные системы и программирование

**Дата:** 2.10.25

**Время проведения:** 3 пара

**Тема:** Команды обмена данными.

**Цель занятия:**

**дидактическая:** изучить команды обмена данными на языке ассемблер

**развивающая**: развивать абстрактное мышление, логику

**Вид занятия** лекция

**Литература**

Юров В.И. Assembler. Учебник для вузов. - 2-ое изд. – СПб.: Питер, 2003, стр. 147.

**Интернет-ресурсы:**

<https://www.sites.google.com/site/sistprogr/lekcii1/lek6>

<https://rutube.ru/video/7845606cfeba7f66eeffb75ee4c57b07/?t=0>

Язык ассемблера для Intel x86 - Урок #3 - Регистры, команда mov, способы адресации

**ЗАДАНИЕ**: законспектировать лекцию с учетом контрольных вопросов.

***КОНСПЕКТ ЛЕКЦИИ***

План

1. Линейные алгоритмы
2. Команды пересылки данных.
3. Работа с адресами и указателями.
4. Работа со стеком.
5. Команды преобразования типов.
6. Команды обработки строк

1. **Линейные алгоритмы**

Последовательность действий, описываемых алгоритмом, может быть:

* *линейной* — все действия выполняются по очереди, друг за другом;
* *нелинейной —* в алгоритме есть точки ветвления, в которых должно приниматься решение о месте, из которого программа продлит свое выполнение, причем решение может носить условный или безусловный характер.

Линейные участки алгоритма обычно содержат команды манипуляции данными, вычисление значений выражений, преобразования данных. В точках разветвления размещают команды сравнения, разных видов перехода, вызова подпрограмм и некоторые другие.

На линейных участках используют следующие группы:

* команды пересылки данных;
* арифметические команды;
* логические команды;
* команды управления состоянием процессора.

Группа команд пересылки данных осуществляют пересылку данных с одного места в другое, запись и чтение информации из портов ввода-вывода, преобразования информации, манипуляции с адресами и указателями, обращения к стеку. Для некоторых из этих команд операция пересылки является только частью алгоритма. Другая его часть выполняет некоторые дополнительные операции над информацией, что пересылается. По функциональному назначению они делятся на команды:

* пересылки данных;
* ввод из порта и вывод в порт;
* работы с адресами и указателями;
* преобразования данных;
* работы со стеком.

**2. Команды пересылки данных**

К группе команд пересылки данных относятся следующие команды:

**Mov**  <операнд назначения>,<операнд-источник>

Данные из А в B:

~~MOV B, A~~

MOV AX, A

MOV B, AX

**xchg** <операнд1>,<операнд2>

Командой MOV нельзя осуществить пересылку из одной области памяти в другую. Если такая необходимость возникает, то нужно использовать в качестве промежуточного буфера любой доступный в данный момент регистр общего назначения.

Например, рассмотрим фрагмент программы для пересылки байта из ячейки fls в ячейку fld:

masm

model small

.data

fls db 5

fld db ?

. code

start:

…

mov al, fIs

mov fld, al

…

end start

*Примечание*:

* Нельзя загрузить в сегментный регистр значения непосредственно из памяти. Для такой загрузки нужен промежуточный объект. Это может быть регистр общего назначения или стек.

fls WORD 5 ; секция данных

…

~~MOV DS, fls~~ ; секция кода

MOV BX, fls

MOV DS, BX

MOV AX, @data

MOV DS, AX

* Нельзя переслать содержимое одного сегментного регистра в другой сегментный регистр. Это объясняется тем, что в системе команд нет соответствующего кода операции. Но необходимость в таком действии часто возникает. Выполнить такую пересылку можно, используя в качестве промежуточных все те же регистры общего назначения. Пример инициализации регистра ES значением из регистра DS:

mov ах, ds

mov es, ах

*Прмечание*:

* Нельзя использовать сегментный регистр CS как операнд назначения.
* ***Длина обоих операндов должна быть одинаковой***. Если она отличается, то в данном случае используется директива указания типа:

*тип* **PTR** [выражение]

Пример:

d\_wrd DWORD 34BAh ; инициализация переменной d\_wrd (4 байта) в секции данных

…

mov al, byte ptr d\_wrd+1 ;пересылка второго байта из двойного слова

Для двунаправленной пересылки данных применяют команду **XCHG**. Для этой операции можно, обычно, применить последовательность из нескольких команд MOV, но из-за того, что операция обмена используется достаточно часто, разработчики системы команд процессора посчитали нужным ввести отдельную команду обмена — XCHG. Естественно, что операнды должны иметь один тип. Не допускается (как и для всех команд ассемблера) напрямую обменивать между собой содержимое двух ячеек памяти. Например:

xchg ах, bx ; обменять содержимое регистров ах и bx

xchg ах, word ptr [si] ; обменять содержимое регистра ах и слова в памяти по адресу в [si]

**3. Работа с адресами и указателями**

При написании программ на ассемблере проводится интенсивная работа с адресами операндов, которые находятся в памяти. Для поддержки такого рода операций есть специальная группа команд, в которую входят следующие команды:

* lea <приемник>,<источник> — загрузка ***эффективного*** адреса;
* lds <приемник>,<источник> — загрузка указателя (сегментная составляющая адреса) в регистр сегмента данных ds, смещение будет помещено в регистр <приемник> ;
* les <приемник>,<источник> — загрузка указателя в регистр дополнительного сегмента данных es;
* lgs <приемник>,<источник> — загрузка указателя в регистр дополнительного сегмента данных gs;
* lfs <приемник>,<источник> — загрузка указателя в регистр дополнительного сегмента данных fs;
* lss <приемник>,<источник> — загрузка указателя в регистр сегмента стека ss.

Если кратко, то эффективный (текущий, действительный, смещение) адрес - это:

БАЗА + СМЕЩЕНИЕ + ИНДЕКС

Команда LEA похожая на команду MOV тем, что она также проводит пересылку, однако команда LEA проводит *пересылку не данных, а эффективного адреса данных* (то есть смещения данных относительно начала сегмента данных) в регистр, указанный операндом <приемник>.

Часто для выполнения некоторых действий в программе недостаточно знать значение одного лишь эффективного адреса данных, а необходимо иметь полный указатель на данные. Полный указатель на данные состоит из сегментной составляющей и смещения. Все остальные команды этой группы позволяет получить в паре регистров такой полный указатель на операнд в памяти. При этом имя сегментного регистра, в который помещается сегментная составляющая адреса, определяется кодом операции. Соответственно, смещения помещается в регистр общего назначения, указанный операндом <приемник>.

Команда LDS REG, MEM выполняет те же действия, что и две следующие команды:

[MOV](https://av-assembler.ru/instructions/mov.php) REG, WORD PTR MEM

MOV DS, WORD PTR MEM+2

Инструкция LDS считывает из памяти по указанному адресу двойное слово (32 бит), содержащее указатель (полный адрес какой-либо ячейки памяти), и загружает младшую половину указателя (то есть относительный адрес, смещение) в указанный в команде регистр общего назначения, а старшую половину указателя (то есть сегментный адрес) в регистр DS.

В качестве первого операнда (REG) команды LDS указывается регистр общего назначения, а в качестве второго (MEM) - ячейка памяти с двухсловным содержимым. Указатель, содержащийся в этой ячейке, может быть адресом как процедуры, так и поля данных.

А теперь ещё раз кратко:

LDS REG, MEM

Эта команда получает двойное слово, адрес которого находится в МЕМ. Затем первые 16 разрядов из этой области памяти загружаются в регистр REG, а вторая часть двойного слова - в регистр DS.

То есть в памяти по адресу MEM содержится указатель (полный адрес), который мы можем разделить на [СЕГМЕНТ:СМЕЩЕНИЕ](https://av-assembler.ru/asm/afd/asm-memory.htm), и с помощью команды LDS записать сегмент в DS, а смещение в указанный в команде регистр (REG).

Необходимость использования команд получения полного указателя данных в памяти, то есть адреса сегмента и значения смещения внутри сегмента, возникает, в частности при работе с цепочками.

**4. Работа со стеком**

Основные причины использования стека в программах:

1. Стек представляет собой очень удобное место для сохранения значения регистров, в том случае, если они используются для нескольких целей. После изменения содержания регистра, его предыдущее значение легко можно восстановить.
2. При выполнении команды CALL процессор сохраняет в стеке адресов следующей за ней команды. Таким образом обеспечивается возврат из процедуры и передача управления следующей за CALL команде.
3. При вызове процедуры ей передается ряд параметров ввода, которые могут быть размещены в стеке.
4. Сразу после вызова внутри процедуры создается ряд локальных переменных, которые также находятся в стеке. Значение этих переменных теряется при возврате управления в программу, что вызывала.

**Команды работы со стеком**

Обычно эти команды работают в паре:

**PUSHx** – сохранить информацию в стеке;

**РОРx** – извлечь информацию из стека.

Стек заполняется в сторону меньших адресов. Поэтому все команды **PUSHx** сначала уменьшают значение указателя стека <SP> = <SP> - 2 (регистр SP всегда показывает на вершину стека), а затем копируют в стек операнды. Команды **РОРХ**, напротив, сначала извлекают нужную информацию в порядке, обратной записи в стек, а затем изменяется содержимое вершины стека: <SP> = <SP> + 2.

**Команды PUSH и POP.** Эти команды запоминают или извлекают 16-разрядный операнд в/из стека. Синтаксис:

**PUSH** Источник

**POP** Приемник

Пример .

Нельзя пересылать сегментные регистры:

**Mov** ES,DS

В этой ситуации можно воспользоваться стеком:

**PUSH** DS

**POP** ES

**Команды PUSHA и РОРA**

Команда PUSHA запоминает все регистры в стеке строго в следующей последовательности: АХ, СХ , DX, ВХ , SP, BP, SІ, DІ. Команда РОРА, напротив, извлекает все эти регистры из стека: DІ, SІ, BP, SP, BX, DX, CX, AX.

Синтаксис:

**PUSHA**

**POPA**

Этим командам операнды не нужны.

**Команды PUSHF и POPF**

Эти команды запоминают (PUSHF) или восстанавливают (POPF) регистр флагов FLAGS, используя стек. Синтаксис:

**PUSHF**

**POPF**

Этим командам операнды тоже не нужны.

**5. Команды преобразования типов**

В операции прибавления должны принимать участие операнды одного формата. Если числа без знака, то в этом случае можно на базе начального операнда сформировать новый операнд (формата двойного слова), старшие разряды которого просто заполнить нулями.

Команды ***преобразования типа -*** расширяют байты в слова, слова — в двойные слова и двойные слова - в учетверенные слова (64-разрядные значения). Команды ***преобразования*** типа особенно полезные при ***преобразовании*** целых со знаком, поскольку они автоматически заполняют старшие биты снова формируемого операнда значениями знакового бита начального объекта. Эта операция приводит к целым значениям того же знака и той же величины, что и начальная, но уже в более длинном формате. Подобное ***преобразование*** называется операцией ***распространения знака***. Существуют два вида команд ***преобразования*** типа.

***Команды без операндов*** *—* эти команды с фиксированными регистрами:

* CBW (Convert Byte to Word) — команда ***преобразования*** байта (в регистре AL) в слово (в регистре АХ) путем распространения значения старшего бита AL на все биты регистра АН;
* CWD (Convert Word to Double) — команда ***преобразования*** слова (в регистре АХ) в двойное слово (в регистрах DX:AX) путем распространения значения старшего бита ах на все биты регистра DX;
* CWDE (Convert Word to Double) — команда ***преобразования*** слова (в регистре АХ) в двойное слово (в регистре ЕАХ) путем распространения значения старшего бита АХ на все биты старшей половины регистра ЕАХ;
* CDQ (Convert Double Word to Quarter Word) — команда ***преобразования*** двойного слова (в регистре ЕАХ) в учетверенное слово (в регистрах EDX:EAX) путем распространения значения старшего бита ЕАХ на все биты регистра EDX;

**6. Команды обработки строк**

* **movsx операнд\_\_1,операнд\_2** — команда пересылки с распространением знака расширяет 8- или 16-разрядное значение операнд\_2, которое может быть регистром или операндом в памяти, до 16- или 32-разрядного значения в одном из регистров, используя знаковый бит для заполнения старших позиций значения операнд\_1 (данную команду удобно использовать для подготовки операндов со знаками до выполнению арифметических операций);
* **movzx операнд\_1,операнд\_2** — команда пересылки с расширением нулем расширяет 8- или 16-разрядное значение операнд\_2 до 16- или 32-разрядного с очисткой (заполнением) нулями старших позиций значения операнд\_2 (данную команду удобно использовать для подготовки операндов без знака к выполнению арифметических действий).

**Контрольные вопросы:**

1. Какие группы команд используют на линейных участках программы в ассемблере?
2. Как делятся команды пересылки данных по функциональному назначению?
3. Назначение и формат команды MOV.
4. Каковы особенности ее использования?
5. Назначение и формат команды XCHG.
6. Какие команды используются для работы с адресами и указателями?
7. В чем заключается разница и сходство команд MOV и LEA?
8. Перечислите основные причины использования стека.
9. Какие команды используются для работы со стеком?
10. Перечислите и опишите команды преобразования типов данных.