Министерство образования Республики Беларусь

«ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. ЕВФРОСИНИИ ПОЛОЦКОЙ»

Факультет информационных технологий

Кафедра технологий программирования

# Методические указания для выполнения лабораторной работы №3 по курсу «Конструирование программного обеспечения»

«Работа со строковыми данными в языке низкого уровня»

Полоцк, 2022 г.

# ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## Программные прерывания

Часто бывает необходимо переслать или сравнить поля данных, которые превышают по длине одно слово.

Например, необходимо сравнить описания или имена для того, чтобы отсортировать их в восходящей последовательности. Элементы такого формата известны как строковые данные и могут являться как символьными, так и числовыми. Для обработки строковых данных Ассемблер имеет пять команд обработки строк:

– ***MOVS*** – переслать один байт или одно слово из одной области памяти в другую;

– ***LODS*** – загрузить из памяти один байт в регистр AL или одно слово в регистр AX;

– ***STOS*** – записать содержимое регистра AL или AX в память;

– ***CMPS*** – сравнить содержимое двух областей памяти, размером в один байт или в одно слово;

– ***SCAS*** – сравнить содержимое регистра AL или AX с содержимым памяти.

Префикс ***REP*** позволяет этим командам обрабатывать строки любой длины.

Цепочечная команда может быть закодирована для повторяющейся обработки одного байта или одного слова за одно выполнение. Например, можно выбрать «байтовую» команду для обработки строки с нечетным числом байт или «двухбайтовую» команду для обработки четного числа байт.

Например, можно кодировать операнды для команды MOVS, но опустить их для MOVSB и MOVSW. Эти команды предполагают, что регистры DI и SI содержат относительные адреса, указывающие на необходимые области памяти (для загрузки можно использовать команду LEA). Регистр SI обычно связан с регистром сегмента данных – DS:SI. Регистр DI всегда связан с регистром дополнительного сегмента – ES:DI. Следовательно, команды MOVS, STOS, CMPS и SCAS требуют инициализации регистра ES (обычно адресом в регистре DS).

## REP: Префикс повторения цепочечной команды

Несмотря на то, что цепочечные команды имеют отношение к одному байту или одному слову, префикс REP обеспечивает повторение команды несколько раз. Префикс кодируется непосредственно перед цепочечной командой, например, REP MOVSB. Для использования префикса REP необходимо установить начальное значение в регистре CX. При выполнении цепочечной команды с префиксом REP происходит уменьшение на 1 значения в регистре CX до нуля.

Таким образом, можно обрабатывать строки любой длины.

Флаг направления определяет направление повторяющейся операции:

– для направления слева направо необходимо с помощью команды CLD установить флаг DF в 0;

– для направления справа налево необходимо с помощью команды STD установить флаг DF в 1.

В следующем примере выполняется пересылка 20 байт из STRING1 в STRING2. Предположим, что оба регистра DS и ES инициализированы адресом сегмента данных:

```
STRING1 DB 20 DUP('*')
STRING2 DB 20 DUP(' ') ...
CLD ;Сброс флага DF
MOV CX,20 ;Счетчик на 20 байт
LEA DI,STRING2 ;Адрес области "куда"
LEA SI,STRING1 ;Адрес области "откуда"
REP MOVSB ;Переслать данные
```

При выполнении команд CMPS и SCAS возможна установка флагов состояния, так чтобы операция могла прекратиться сразу после обнаружения необходимого условия. Ниже приведены модификации префикса REP для этих целей:

– REP – повторять операцию, пока CX не равно 0;

– REPZ или REPE – повторять операцию, пока флаг ZF показывает «равно или ноль».

– Прекратить операцию при флаге ZF, указывающему на не равно или не ноль или при CX равном 0;

– REPNE или REPNZ – повторять операцию, пока флаг ZF показывает «не равно или не ноль».

– Прекратить операцию при флаге ZF, указывающему на «равно или нуль» или при CX равным 0.

## MOVS: Пересылка строк

Команда MOVS с префиксом REP и длиной в регистре CX может выполнять пересылку любого числа символов. Для области, принимающей строку, сегментным регистром, является регистр ES, а регистр DI содержит относительный адрес области, передающей строку. Сегментным регистром является регистр DS, а регистр SI содержит относительный адрес. Таким образом, в начале программы перед выполнением команды MOVS необходимо инициализировать регистр ES вместе с регистром DS, а также загрузить требуемые относительные адреса полей в регистры DI и SI.

В зависимости от состояния флага DF команда MOVS производит увеличение или уменьшение на 1 (для байта) или на 2 (для слова) содержимого регистров DI и SI. Приведем команды, эквивалентные цепочечной команде REP MOVSB:

```
JCXZ LABEL2
LABEL1: MOV AL,[SI]
MOV [DI],AL
INC/DEC DI ;Инкремент или декремент
UNC/DEC SI ;Инкремент или декремент
LOOP LABEL1
LABEL2: ...
```

## LODS: Загрузка строки

Команда LODS загружает из памяти в регистр AL один байт или в регистр AX одно слово. Адрес памяти определяется регистрами DS:SI. В зависимости от значения флага DF происходит увеличение или уменьшение регистра SI.

Поскольку одна команда LODS загружает регистр, то практической пользы от префикса REP в данном случае нет. Часто простая команда MOV полностью адекватна команде LODS, хотя MOV генерирует три байта машинного кода, а LODS – только один, но требует инициализацию регистра SI. Можно использовать команду LODS в том случае, когда требуется продвигаться вдоль строки (по байту или по слову), проверяя загружаемый регистр на конкретное значение.

Команды, эквивалентные команде LODSB:

```
MOV AL,[SI]
INC SI
```

## STOS: Запись строки

Команда STOS записывает (сохраняет) содержимое регистра AL или AX в байте или в слове памяти. Адрес памяти всегда представляется регистрами ES:DI. В зависимости от флага DF команда STOS также увеличивает или уменьшает адрес в регистре DI на 1 для байта или на 2 для слова.

Практическая польза команды STOS с префиксом REP – инициализация области данных конкретным значением, например, очистка дисплейного буфера пробелами. Длина области (в байтах или в словах) загружается в регистр AX.

Команды, эквивалентные команде REP STOSB:

```
JCXZ LABEL2
LABEL1: MOV [DI],AL
INC/DEC DI ;Инкремент или декремент
```

```
LOOP LABEL1
LABEL2: ...
```

## CMPS: Сравнение строк

Команда CMPS сравнивает содержимое одной области памяти (адресуемой регистрами DS:SI) с содержимыми другой области (адресуемой как ES:DI). В зависимости от флага DF команда CMPS также увеличивает или уменьшает адреса в регистрах SI и DI на 1 для байта или на 2 для слова. Команда CMPS устанавливает флаги AF, CF, OF, PF, SF и ZF. При использовании префикса REP в регистре CX должна находиться длина сравниваемых полей. Команда CMPS может сравнивать любое число байт или слов.

Рассмотрим процесс сравнения двух строк, содержащих имена JEAN и JOAN. Сравнение побайтно слева направо приводит к следующему:

J : J Равно
E : O Не равно (E меньше O)
A : A Равно
N : N Равно

Сравнение всех четырех байт заканчивается сравнением N:N – равно/нуль. Так как имена «не равны», операция должна прекратиться, как только будет обнаружено условие «не равно».

Для этих целей команда REP имеет модификацию REPE, которая повторяет сравнение до тех пор, пока сравниваемые элементы равны, или регистр CX не равен нулю. Кодируется повторяющееся однобайтовое сравнение следующим образом:

```
REPE CMPSB
```

## SCAS: Сканирование строк

Команда SCAS отличается от команды CMPS тем, что сканирует (просматривает) строку на определенное значение байта или слова. Команда SCAS сравнивает содержимое области памяти (адресуемой регистрами ES:DI) с содержимым регистра AL или AX. В зависимости от значения флага DF команда SCAS также увеличивает или уменьшает адрес в регистре DI на 1 для байта или на 2 для слова. Команда SCAS устанавливает флаги AF, CF, OF, PF, SF и ZF. При использовании префикса REP и значения длины в регистре CX команда SCAS может сканировать строки любой длины.

Команда SCAS особенно полезна, например, в текстовых редакторах, где программа должна сканировать строки, выполняя поиск знаков пунктуации: точек, запятых и пробелов.

Команда SCASW сканирует в памяти слово на соответствие значению в регистре AX. При использовании команд LODSW или MOV для пересылки слова в

регистр AX, следует помнить, что первый байт будет в регистре AL, а второй байт – в регистре AH. Так как команда SCAS сравнивает байты в обратной последовательности, то операция корректна.

## Сканирование и замена

В процессе обработки текстовой информации может возникнуть необходимость замены определенных символов в тексте на другие, например, подстановка пробелов вместо различных редактирующих символов. В приведенном ниже фрагменте программы осуществляется сканирование строки STRING и замена символа амперсанд (&) на символ пробела.

Когда команда SCASB обнаружит символ & (в примере это будет позиция STRING+8), то операция сканирования прекратится и регистр DI будет содержать адрес STRING+9. Для получения адреса символа & необходимо уменьшить содержимое DI на единицу и записать по полученному адресу символ пробела.

```
STRLEN EQU 15 ;Длина поля
STRING STRING DB 'The time&is now' ...
CLD MOV AL,'&' ;Искомый символ
MOV CX,STRLEN ;Длина поля
STRING LEA DI,STRING ;Адрес поля
STRING REPNE SCASB ;Сканировать
JNZ K20 ;Символ найден?
DEC DI ;Да – уменьшить адрес
MOV BYTE PTR[DI],20H ;Подставить пробел
K20: RET
```

## Альтернативное кодирование

При использовании команд MOVSB или MOVSW Ассемблер предполагает наличие корректной длины строковых данных и не требует кодирования операндов в команде. Для команды MOVS длина должна быть закодирована в операндах. Например, если поля FLDA и FLDB определены как байтовые (DB), то команда REP MOVS FLDA,FLDB предполагает повторяющуюся пересылку байтов из поля FLDB в поле FLDA. Эту команду можно записать также в следующем виде:

REP MOVS ES:BYTE PTR[DI],DS:[SI]

Однако загрузка регистров DI и SI адресами FLDA и FLDB обязательна в любом случае.

## Дублирование образца

Команда STOS бывает полезна для установки в некоторой области определённых значений байтов и слов. Для дублирования образца, длина которого превышает размер слова, можно использовать команду MOVS с небольшой модификацией. Предположим, что необходимо сформировать строку следующего вида:

```
***---***---***---***---***---
...
```

Вместо того, чтобы определять полностью всю строку, можно определить только первые шесть байтов. Закодируем образец непосредственно перед обрабатываемой строкой следующим образом:

```
PATTERN DB '***---'
DISAREA DB 42 DUP(?)
.
.
CLD
MOV CX,21
LEA DI,DISAREA
LEA SI,PATTERN
REP MOVSW
```

В процессе выполнения команда MOVSW сначала пересылает первое слово (**) из образца PATTERN в первое слово области DISAREA, затем – второе слово (*-), потом третье (--).

К этому моменту регистр DI будет содержать адрес DISAREA+6, а регистр SI – PATTERN+6, который также является адресом DISAREA. Затем команда MOVSW автоматически дублирует образец, пересылая первое слово из DISAREA в DISAREA+6, из DISAREA+2, в DISAREA+8, из DISAREA+4 в DISAREA+10 и так далее. В результате образец будет полностью продублирован по всей области DISAREA.

Данную технику можно использовать для дублирования в области памяти любого образца любой длины. Образец должен быть расположен непосредственно перед принимающей областью.

***Важно:***

– Для цепочечных команд MOVS, STOS, CMPS и SCAS не забывайте инициализировать регистр ES.

– Сбрасывайте (CLD) или устанавливайте (STD) флаг направления в соответствии с направлением обработки.

– Не забывайте устанавливать в регистрах DI и SI необходимые значения. Например, команда MOVS предполагает операнды DI,SI, а команда CMPS – SI,DI.

– Инициализируйте регистр CX в соответствии с количеством байтов или слов, участвующих в процессе обработки.

– Для обычной обработки используйте префикс REP для команд MOVS и STOS и модифицированный префикс (REPE или REPNE) для команд CMPS и SCAS.

– Помните об обратной последовательности байтов в сравниваемых словах при выполнении команд CMPSW и SCASW.

– При обработке справа налево устанавливайте начальные адреса на последний байт обрабатываемой области. В случае, если, например, поле NAME1 имеет длину 10 байтов, то для побайтовой обработки данных в этой области справа налево начальный адрес, загружаемый командой LEA, должен быть NAME1+9. Для обработки слов начальный адрес в этом случае – NAME1+8.

## Обработка таблиц. Определение таблиц

Многие программные применения используют табличную организацию таких данных, как имена, описания, размеры, цены. Определение и использование таблиц включает одну новую команду Ассемблера – ***XLAT***.

Организация поиска в таблице зависит от способа ее определения. Существует много различных вариантов определения таблиц и алгоритмов поиска.

Для облегчения табличного поиска большинство таблиц определяются систематично, то есть, элементы таблицы имеют одинаковый формат (символьный или числовой), одинаковую длину и восходящую или нисходящую последовательность элементов.

Возьмем, к примеру, стек, представляющий собой таблицу из 64-х неинициализированных слов:

```
STACK DW 64 DUP(?)
```

Следующие две таблицы инициализированы символьными и числовыми значениями:

```
MONTAB DB 'JAN','FEB','MAR', ... ,'DEC'
COSTAB DB 205,208,209,212,215,224,...
```

Таблица ***MONTAB*** определяет алфавитные аббревиатуры месяцев, а ***COSTAB*** – определяет таблицу номеров служащих. Таблица может также содержать смешанные данные (регулярно чередующиеся числовые и символьные поля). В следующей ассортиментной таблице каждый числовой элемент (инвентарный номер) имеет две цифры (один байт), а каждый символьный элемент (наименование) имеет девять байтов. Точки, показанные в наименовании «Paper» дополняют длину этого поля до 9 байт. Точки показывают, что недостающее пространство должно присутствовать. Вводить точки необязательно.

```
STOKTBL DB 12,'Computers',14,'Paper....',17,'Diskettes'
```

Для ясности можно закодировать элементы таблицы вертикально:

```
STOKTBL DB 12, 'Computers' DB 14, 'Paper....' DB 17, 'Diskettes'
```

Рассмотрим теперь различные способы использования таблиц в программах.

## Прямой табличный доступ

Предположим, что пользователь ввел номер месяца – 03 и программа должна преобразовать этот номер в алфавитное значение March. Программа для выполнения такого преобразования включает определение таблицы алфавитных названий месяцев, имеющих одинаковую длину. Так как самое длинное название – September, то таблица имеет следующий вид:

```
MONTBL DB 'January..' DB 'February.' DB 'March....'
```

Каждый элемент таблицы имеет длину 9 байт. Адрес элемента 'January' – MONTBL+0, 'February' – MONTBL+9, 'March' – MONTBL+18. Для локализации месяца 03, программа должна выполнить следующее:

1. Преобразовать введенный номер месяца из ASCII 33 в двоичное 03.
2. Вычесть единицу из номера месяца: 03 – 1 = 02 3. Умножить результат на длину элемента (9): 02 x 9 = 18 4. Прибавить произведение (18) к адресу MONTBL; в результате получится адрес требуемого названия месяца: MONTBL+18.

Описанная техника работы с таблицей называется прямым табличным доступом. Поскольку данный алгоритм непосредственно вычисляет адрес необходимого элемента в таблице, то в программе не требуется выполнять операции поиска.

Хотя прямая табличная адресация очень эффективна, она возможна только при последовательной организации. То есть можно использовать такие таблицы, если элементы располагаются в регулярной последовательности: 1, 2, 3,... или 106, 107, 108,... или даже 5, 10, 15. Однако, не всегда таблицы построены таким образом.

## Табличный поиск

Некоторые таблицы состоят из чисел, не имеющих видимой закономерности. Характерный пример – таблица инвентарных номеров с последовательными номерами, например, 134, 138, 141, 239 и 245. Другой тип таблиц состоит из распределенных по ранжиру величин, таких как подоходный налог.

## Таблицы с уникальными элементами

Инвентарные номера большинства фирм часто не имеют последовательного порядка. Номера, обычно, группируются по категориям, первые цифры указывают на мебель или приборы, или номер отдела. Кроме того, время от времени номера удаляются, а новые добавляются. В таблице необходимо связать инвентарные номера и их конкретные наименования (и, если требуется, включить стоимость). Инвентарные номера и наименования могут быть определены в различных таблицах, например:

```
STOKNOS DB '101','107','109',...
STOKDCR DB 'Excavators','Processors','Assemblers',...
```

или в одной таблице, например:

```
STOKTAB DB '101','Excavators' DB '107','Processors'  DB
'109','Assemblers' ...
```

## Таблицы с ранжированием

Подоходный налог дает характерный пример таблицы с ранжированными значениями. Представим себе таблицу, содержащую размеры доходов облагаемых налогами, процент налога и поправочный коэффициент:

В налоговой таблице процент увеличивается в соответствии с увеличением налогооблагаемого дохода. Элементы таблицы доходов содержат максимальные величины для каждого шага:

```
TAXTBL DD 100000,250000,425000,600000,999999
```

Для организации поиска в такой таблице, программа сравнивает доход налогоплательщика с табличным значением дохода:

– если меньше или равно, то использовать соответствующий процент и поправку;

– если больше, то перейти к следующему элементу таблицы.

## Таблицы с элементами переменной длины

Существуют таблицы, в которых элементы имеют переменную длину. Каждый элемент такой таблицы может завершаться специальным символом ограничителем, например, шест.00; конец таблицы можно обозначить ограничителем шест.FF. В этом случае необходимо гарантировать, чтобы внутри элементов таблицы не встречались

указанные ограничители. Помните, что двоичные числа могут выражаться любыми битовыми комбинациями. Для поиска можно использовать команду SCAS.

## Транслирующая команда XLAT

Команда XLAT транслирует содержимое одного байта в другое предопределенное значение. С помощью команды XLAT можно проверить корректность содержимого элементов данных. При передаче данных между персональным компьютером и ЕС ЭВМ (IBM) с помощью команды XLAT можно выполнить перекодировку данных между форматами ASCII и EBCDIC.

В следующем примере происходит преобразование цифр от 0 до 9 из кода ASCII в код EBCDIC. Так как представление цифр в ASCII выглядит как шест.30-39, а в EBCDIC – шест.F0-F9, то замену можно выполнить командой OR. Однако, дополнительно преобразуем все остальные коды ASCII в пробел (шест.40) в коде EBCDIC. Для команды XLAT необходимо определить таблицу перекодировки, которая учитывает все 256 возможных символов, с кодами EBCDIC в ASCII позициях:

```
XLTBL DB 47 DUP(40H) ;Пробелы в коде EBCDIC
DB 0F0H,0F1H,0F2H,0F3H,...,0F9H ;0-9 (EBCDIC)
DB 199 DUP(40H) ;Пробелы в коде EBCDIC
```

Команда XLAT предполагает адрес таблицы в регистре BX, а транслируемый байт (например, поля ASCNO) в регистре AL. Следующие команды выполняют подготовку и трансляцию байта:

```
LEA BX,XLTBL
MOV AL,ASCNO
XLAT
```

Команда XLAT использует значение в регистре AL в качестве относительного адреса в таблице, то есть, складывает адрес в BX и смещение в AL. В случае, если, например, ASCNO содержит 00, то адрес байта в таблице будет XLTBL+00 и команда XLAT заменит 00 на шест.40 из таблицы. В случае, если поле ASCNO содержит шест.32, то адрес соответствующего байта в таблице будет XLTBL+50. Этот байт содержит шест.F2 (2 в коде EBCDIC), который команда XLAT загружает в регистр AL.

## Операторы типа, длина и размеры

Ассемблер содержит ряд специальных операторов, которые могут оказаться полезными при программировании. Например, при изменении длины таблицы придется модифицировать программу (для нового определения таблицы) и

процедуры, проверяющие конец таблицы. В этом случае использование операторов ***TYPE (тип)***, ***LENGTH (длина)*** и ***SIZE (размер)*** позволяют уменьшить число модифицируемых команд.

Рассмотрим определение следующей таблицы из десяти слов:

```
TABLEX DW 10 DUP(?) ;Таблица из 10 слов
```

Программа может использовать оператор TYPE для определения типа (DW в данном случае), оператор LENGTH для определения DUP-фактора (10) и оператор SIZE для определения числа байтов (10 x 2 = 20). Следующие команды иллюстрируют три таких применения:

```
MOV AX,TYPE
TABLEX ;AX=0002
MOV BX,LENGTH
TABLEX ;BX=000A (10)
MOV CX,SIZE
TABLEX ;CX=0014 (20)
```

Значения LENGTH и SIZE можно использовать для окончания табличного поиска или сортировки. Например, если регистр SI содержит продвинутый адрес таблицы при осуществлении поиска, то проверка на конец таблицы может быть следующий:

```
CMP SI,SIZE
TABLEX
```

## Порядок выполнения работы

**Задание**. Написать программу со следующим алгоритмом:

– вывести на экран приглашение, и ввести с клавиатуры символьную подстроку для поиска;

– вывести на экран еще одно приглашение, и ввести с клавиатуры символьную строку с именем файла;

– открыть файл, прочитать его в буфер и найти в содержимом подстроку, используя строковые инструкции;

– вывести результат поиска в виде сообщения «Найдено» или «Не найдено»;

– завершить программу.

## Содержание отчета

Отчет должен включать:

а) титульный лист;

б) формулировку цели работы;

в) описание результатов выполнения задания:

– листинги программ;

– результаты выполнения программ;

г) выводы, согласованные с целью работы.

## Контрольные вопросы

1. Каковы особенности строковых инструкций?

2. Для чего используется флаговый регистр в строковых инструкциях?

3. Каково назначение и механизм применения префикса повторения строкового примитива?

4. Что возвращается в случае ошибки открытия файла?