**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Организация систем и ЭВМ»**

Тема: Написание собственного прерывания и работа со строками

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Еникеев А.А. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2024

## Цель работы

Изучить прерывания и работу со строками на языке ассемблера. Написать собственное прерывание и реализовать обработку строки.

## Задание

Состоит из двух основных задач:

1) Реализация сценария работы с прерываниями в соответствии с вариантом.

2) Реализация преобразования строки с использованием команд работы со строками в соответствии с вариантом.

|  |  |
| --- | --- |
| N вар. | Ширф задания |
| 9 | 9з |

Сценарии работы с прерываниями.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № сц. | Действия основной программы | Действия 1сh | Действия 9h | Действия 4bh |
| 9 | 2) замена 9h;  3) замена 1ch;  4) ожидание ввода строки. | после замены через 6 сек. восст. 9h и себя. | 1) вып. работу;  2) восст. себя. |  |

з) Вывести количество случаев, когда во входной строке два последовательных символа одинаковы.

Замечания:

1) В сценариях опущены:

– Вывод приветственного сообщения и ввод строки пользователем. Это 1-ый пункт основной программы каждого сценария (поэтому основная программа во всех сценариях начинается с пункта 2). Слова́ «Ожидание ввода строки», которые встречаются в таблице сценариев, – это всего лишь организация задержки с ожиданием нажатия клавиш, для ввода данных не используется.

– Действия по восстановлению изменённых прерываний, если они не требуются по сценарию специально. После завершения программы все изменённые прерывания всегда должны быть восстановлены (независимо от сценария). Перед завершением программы следует восстановить в том числе и те вектора прерываний, которые могли бы восстановиться по таймеру. После завершения программы DOS освобождает память, занимаемую программой, что может помешать корректному восстановлению векторов. Поэтому рассчитывать на восстановление по таймеру после завершения программы будет неправильно.

2) «Ожидание ввода строки» следует делать так, чтобы пользователь мог нажать множество клавиш, прежде чем решит завершить ввод нажатием Enter.

3) Для 1ch описан не алгоритм работы одного вызова, а последовательность действий, выполняемая в результате множества вызовов. Время должно измеряться через подсчёт количества выполненных вызовов, функцию задержки использовать не следует.

4) «вып. работу» означает, что надо на основе введённой строки (п. 1 основной программы) создать модифицированную строку и вывести её на экран. При выполнении преобразования нельзя портить исходную строку, результат преобразования должен записываться в выходную строку.

5) Перед запуском ожидания нажатия клавиши («ожидание ввода строки» в таблице) вывести сообщение об этом.

6) Перед заменой 9h следует сделать небольшую задержку (см. 3.2 «Задержка во времени»), чтобы предшествующая активность пользователя была обработана до того, как 9h будет изменён;

7) Для исключения возможного взаимного влияния системных и пользовательских прерываний рекомендуется отвести в программе под стек не менее 1Кбайт.

## Основные теоритические положения

Прерывание – это процесс вызова процедур для выполнения некоторой задачи, обычно связанной с обслуживанием некоторых устройств (обработка сигнала таймера, нажатия клавиши и т.д.). Когда возникает прерывание, процессор прекращает выполнение текущей программы (если её приоритет ниже) и запоминает в стеке вместе с регистром флагов адрес возврата (CS:IP) – ме́ста, с которого будет продолжена прерванная программа. Затем в CS:IP загружается адрес программы обработки прерывания и ей передаётся управление.

Адреса 256 программ обработки прерываний, так называемые векторы прерывания, имеют длину по 4 байта (в первых двух хранится значение IP, во вторых – CS) и хранятся в младших 1024 байтах памяти.

Программа обработки прерывания должна заканчиваться инструкцией IRET (возврат из прерывания), по которой из стека восстанавливается адрес возврата и регистр флагов.

Программа, использующая новые программы обработки прерываний, при своём завершении должна восстанавливать оригинальные векторы прерываний. Функция 35 прерывания 21H возвращает текущее значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX.

Для задания адреса собственного прерывания с заданным номером в таблицу векторов прерываний используется функция 25H прерывания 21H, которая устанавливает вектор прерывания на указанный адрес нового обработчика.

При использовании функции 25H прерывания 21H DOS знает, что вы делаете, и гарантирует, что в момент записи прерывания будут заблокированы. Поэтому вызывать команды CLI и STI не нужно. Но они понадобятся в случае ручного изменения вектора прерывания (т.е. без вызова INT 21H), чтобы не допустить возможного возникновения ужасной ситуации, когда сегмент был переопределён, а смещение осталось старым (или наоборот).

Прерывания бывают аппаратные (вызываемые в результате сигналов от оборудования) и программные (вызываемые в коде). В лабораторной работе предлагаются к замене вектора́ следующих прерываний:

– 1CH и 08H – можно рассматривать их как аппаратные прерывания, генерируемые системным таймером; генерируются автоматически 18,2 раза в сек.;

– 09H – аппаратное прерывание, возникающее в результате нажатия клавиш клавиатуры;

Для обработки строковых данных ассемблер имеет пять групп 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.

Каждая команда имеет модификации, указывающие размер операнда: байт (B), слово (W), двойное слово (D). Например: MOVSB, MOVSW, MOVSD.

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).

Префиксы REP/REPE/REPZ/REPNE/REPNZ позволяют этим командам обрабатывать строки любой длины.

## Выполнение работы

1. Основаная часть программы (процедура MAIN)

- Вывод приветственного сообщения: для вывода приветственной строки используется процедура WriteMSG, которая вызвает прерывание 21h с регистром AH = 9h.

- Ввод строки пользователем: после привественного сообщения программа ожидает ввода строки пользователем, для чтения используется процедура StringRead.

- Сохранение вектора оригинального обработчика 9h: вызывается прерывание 21h с AH = 35h и AL = 9h, которое сохранение сегмент и смещение вектора 9h

- Задержка перед заменой оригинального обработчика 9h: используется прерывание 15h с AH = 86h, которое выполняет ожидание в течении почти 1 секунды.

- Замена обработчика 9h: устанавливается новый обработчик 9h — NEW\_INT\_9H (описание реализации см. ниже).

- Сохранени вектора оригинального обработчика 1ch: аналогично с сохранением обработчика 9h, только AL = 1ch.

- Замена обработчика 1сh: устанавливается новый обработчик 1сh — NEW\_INT\_1CH (описание реализации см. ниже).

- Выводиться подсказка: программа ожидает нажатия любой клавиши для вызова прерывания 9h, пока не нажата клавиша „Enter’’.

- После нажатия „Enter’’ программа восстанавливает оригинальные обработчики 1ch и 9h, завершает своё выполнение.

1. Этапы работы переопределенного обработчика 1ch (NEW\_INT\_1CH)

- Данное прерыание генерируется автоматически программой каждые 18,2 сек., с каждым вызовом увеличивается счетчик запусков TIMER\_COUNT.

- Если количество вызовов равно 109, значит прошло 6 секунд., в этом случае обработчик восстанавливает себя и 9h до оригинального вызовом процедуры RestoreInterrupts

- Если количество вызовов не достигло 109, выполняет возврат и прерывания.

1. Этапы работы переопределенного обработчика 9h (NEW\_INT\_9H)

- Вызывает процедуру ProcessString (описание см. ниже), которая обрабатывает строку и выводит результат.

- Восстанавливает себя и переходит к вызову оригинального обработчика 9h, чтобы обработка сигналов клавиатуры не была нарушена.

1. Этапы работы процедуры для обработки строки (ProcessString)

- Инициализация переменных:

- Устанавливается длина входной строки из STR1HEAD[1] в CL.

- Регистр BX обнуляется для подсчёта повторяющихся символов.

- Указатели SI и DI настраиваются для строк STR1 и STR2.

- Чтение первого символа строки: Загружается первый символ строки STR1 в AL.

- Цикл обработки символов (NEXT\_CHAR):

- Проверяется длина строки: если CL == 0, обработка завершена.

- Сравниваются текущий и следующий символы.

- Если символы совпадают, увеличивается счётчик (BX).

- Если нет, переход к следующему символу.

- Завершение обработки строки (END\_PROCESS):

- Загружается результат подсчёта повторов (BX) в AX.

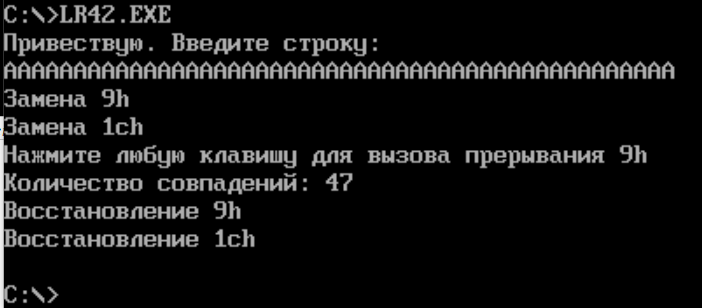
- Вызывается процедура AddNumberToSTR2, чтобы преобразовать число в строку и добавить в STR2.

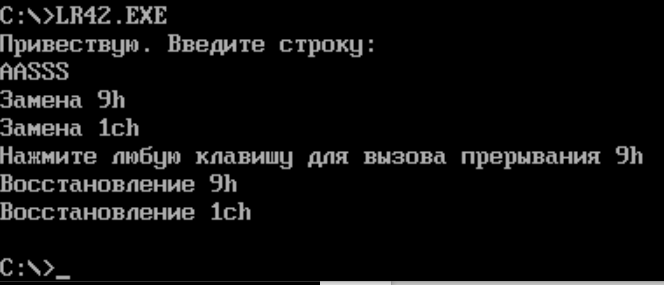
- Добавление символа $ в конец строки: завершается строка STR2 для правильного отображения.

- Вывод результата на экран: Выводится строка RESULT с результатом обработки

**Тестирование**

Тестирование сценария работы с прерываниями см. на рис. 1 и 2.

  
Рисунок 1 - При нажатии любой клавиши в течении 6 секунд после подсказки

  
Рисунок 2 - Бездействие в течении 6 секунд после вывода подсказки

Тестирование обработки строки см. в табл. 1.

Табл. 1.

|  |  |  |
| --- | --- | --- |
| Введенная строка | Полученный результат | Ожидаемый результат |
| A | 0 | 0 |
| AAANNNMKK | 5 | 5 |
| $ | 0 | 0 |
| SSSSSSSSSS | 9 | 9 |

**Выводы**

Были изучены прерывания и работа со строками на языке Ассемлер. Разработана и протестирована программа, где релизованы собственные прерывания и обработка строки.

# Приложение A

**Исходный код программы**

**Файл LR4.ASM:**

.MODEL SMALL

.STACK 1024h

.DATA

KEEP\_OLD\_9H DW 0, 0 ; Хранение старого обработчика 9h

KEEP\_OLD\_1CH DW 0, 0 ; Хранение старого обработчика 1Ch

EOFLine EQU '$' ;Определение символьной константы "Конец строки"

EmptyLine DB 0Dh, 0Ah, '$' ; Пустая строка с переходом на новую строку

Replacement\_1ch DB 'Замена 1ch', 0ah, 0dh, EOFLine

Recovery\_1ch DB 'Восстановление 1ch', 0ah, 0dh, EOFLine

Replacement\_9h DB 'Замена 9h', 0ah, 0dh, EOFLine

Recovery\_9h DB 'Восстановление 9h', 0ah, 0dh, EOFLine

WELCOME\_MSG DB 'Привествую. Введите строку:', 0ah, 0dh, EOFLine ; Привественное сообщение

PROMPT\_MSG DB 'Нажмите любую клавишу для вызова прерывания 9h', 0ah, 0dh, EOFLine ; Подсказка

RESULT DB "Количество совпадений: ", EOFLine

STR1HEAD DB 50h, 0 ; Заголовок строки (максимум 80 символов)

STR1 DB 80 DUP('\*'), 0ah, 0dh, EOFLine ; Буфер строки

STR2 DB 5 DUP(' '), 0ah, 0dh, EOFLine ; Выходной буфер

TIMER\_COUNT DW 0 ; Счётчик вызовов 1Ch

.CODE

MAIN PROC FAR ; Основная часть программы

MOV AX, @data ; Инициализация сегмента данных

MOV DS, AX

; Вывод приветствия

MOV DX, OFFSET WELCOME\_MSG

CALL WriteMSG

; Ввод строки пользователем

MOV DX, OFFSET STR1HEAD

CALL StringRead

; Сохранение оригинального вектора 9h

MOV AH, 35h ; Функция получения вектора

MOV AL, 9h ; Номер вектора (прерывания)

INT 21h ; Вызов прерывания

MOV KEEP\_OLD\_9H+2, ES ; Сохранение сегмента вектора

MOV KEEP\_OLD\_9H, BX ; Сохранение смещения вектора

; Задержка перед заменой 9h

MOV CX, 0eh ; 14 \* 65535 мкс = 917490 мкс (~0.917 сек)

MOV DX, 0ffffh ; Дополнительные 65535 мкс (~0.0655 сек)

MOV AH, 86h ; Функция "ждать"

INT 15h ; Вызов задержки (~0.983 секунд)

; Вывод пустой строки

LEA DX, EmptyLine

CALL WriteMSG

; Вывод сообщения о замене

LEA DX, Replacement\_9h

CALL WriteMSG

; Замена вектора 9h

PUSH DS ; Сохраняем текущий сегмент данных в стеке

MOV DX, OFFSET NEW\_INT\_9H ; Заносим смещение нового обработчика прерывания 9h в DX

MOV AX, SEG NEW\_INT\_9H ; Заносим сегмент нового обработчика в AX

MOV DS, AX ; Устанавливаем DS для доступа к сегменту обработчика

MOV AH, 25h ; Выбираем функцию 25h DOS (установить вектор прерывания)

MOV AL, 9h ; Указываем номер прерывания (9h - клавиатурное)

INT 21h ; Вызов DOS для замены вектора

POP DS ; Восстанавливаем исходный сегмент данных из стека

; Сохранение оригинального вектора 1Ch

MOV AH, 35h ; Функция получения вектора

MOV AL, 1Ch ; Номер вектора (прерывания)

INT 21h ; Вызов прерывания

MOV KEEP\_OLD\_1CH+2, ES ; Сохранение сегмента вектора

MOV KEEP\_OLD\_1CH, BX ; Сохранение смещения вектора

; Замена вектора 1Ch

MOV DX, OFFSET Replacement\_1ch

CALL WriteMSG

PUSH DS ; Сохраняем текущий сегмент данных в стеке

MOV DX, OFFSET NEW\_INT\_1CH ; Заносим смещение нового обработчика прерывания 1Ch в DX

MOV AX, SEG NEW\_INT\_1CH ; Заносим сегмент нового обработчика в AX

MOV DS, AX ; Устанавливаем DS для доступа к сегменту обработчика

MOV AH, 25h ; Выбираем функцию 25h DOS (установить вектор прерывания)

MOV AL, 1Ch ; Указываем номер прерывания (1Ch - аппаратное прерывание от таймера)

INT 21h ; Вызов DOS для замены вектора

POP DS ; Восстанавливаем исходный сегмент данных из стека

; Вывод подсказки

LEA DX, PROMPT\_MSG

CALL WriteMSG

; Ожидание нажатия "Enter"

WAIT\_FOR\_ENTER:

MOV AH, 0h ; Функция "проверка клавиши" (0h) - проверка, была ли нажата клавиша

INT 16h ; Вызываем прерывание для чтения клавиши (результат в AL)

CMP AL, 0Dh ; Сравниваем содержимое AL (код клавиши) с кодом Enter (0Dh)

JNE WAIT\_FOR\_ENTER ; Если это не клавиша Enter, продолжаем ожидание

DONE:

; Восстановление 1ch и 9ch

;CALL ProcessString

CALL RestoreInterrupts

; Завершение программы

MOV AH, 4Ch

INT 21h

MAIN ENDP

StringRead PROC ; Чтение введенной строки

; Сохраним регистры, значение которых будет изменено

PUSH AX

PUSH BP

PUSH BX

MOV AH,0ah ; Функция ввода строки

PUSH DX ; Смещение заголовка строки...

INT 21h ; Вызов функции DOS ввода строки

POP BP ; ...поместить в bp

XOR BX,BX

MOV BL,DS:[BP+1] ; Теперь в bx количество введённых символов

ADD BX,BP ; Теперь bx указывает на конечный введённый символ

ADD BX,2 ; Теперь bx указывает на байт, следующий за финальным 0dh

MOV WORD PTR [BX+1],240ah ; Добавить в конец 0ah и '$'

; Возвращение сохраненных регистров

POP BX

POP BP

POP AX

RET

StringRead ENDP

WriteMSG PROC

PUSH AX ; Сохраняем значение регистра

MOV AH, 9h

INT 21h

POP AX ; Восстанавливаем регистр AX

RET

WriteMSG ENDP

RestoreInterrupts PROC

PUSH AX

MOV DX, OFFSET Recovery\_9h

CALL WriteMSG

MOV DX, OFFSET Recovery\_1ch

CALL WriteMSG

MOV BYTE PTR [Recovery\_9h], '$'

MOV BYTE PTR [Recovery\_1ch], '$'

; Восстановление 9h

PUSH DS ; Сохраняем текущий сегмент данных (DS) на стеке

MOV DX, KEEP\_OLD\_9H ; Смещение для оргинального обработчика 9h

MOV AX, KEEP\_OLD\_9H+2 ; Сегмент для оргинального обработчика 9h

MOV DS, AX ; Загружаем сегмент обработчика 9h в регистр DS

MOV AH, 25h ; Функция 25h прерывания 21h для замены вектора прерывания

MOV AL, 9h ; Указываем, что восстанавливаем обработчик для прерывания 9h

INT 21h ; Выполняем прерывание 21h для восстановления обработчика 9h

POP DS ; Восстанавливаем исходное значение сегмента данных (DS)

; Восстановление 1Ch

PUSH DS ; Сохраняем текущий сегмент данных (DS) на стеке

MOV DX, KEEP\_OLD\_1CH ; Смещение для оргинального обработчика 1ch

MOV AX, KEEP\_OLD\_1CH+2 ; Сегмент для оргинального обработчика 1ch

MOV DS, AX ; Загружаем сегмент обработчика 1ch в регистр DS

MOV AH, 25h ; Функция 25h прерывания 21h для замены вектора прерывания

MOV AL, 1Ch ; Указываем, что восстанавливаем обработчик для прерывания 1ch

INT 21h ; Выполняем прерывание 21h для восстановления обработчика 1ch

POP DS ; Восстанавливаем исходное значение сегмента данных (DS)

POP AX

RET

RestoreInterrupts ENDP

NEW\_INT\_1CH PROC ; Новый обработчик 1ch

; Сохраняем регистры, которые изменяеются в RestoreInterrupts

PUSH AX

PUSH DX

; Увеличиваем счетчик запусков

INC TIMER\_COUNT

; Сравниваем текущее значение счетчика с 109 (примерно 6 секунд)

CMP TIMER\_COUNT, 109

; Если счетчик меньше 109, переходим к метке CONTINUE, не делая ничего

JNE CONTINUE

; Иначе вызываем функцию восстановления старых векторов прерываний (в том числе для 9h)

CALL RestoreInterrupts

CONTINUE:

; Восстанавливаем значения регистров, которые были сохранены в начале процедуры

POP DX

POP AX

; Возврат из прерывания с помощью IRET

IRET

NEW\_INT\_1CH ENDP

NEW\_INT\_9H PROC ; Новый обработчик 9h

; Сохранение регистров

PUSH AX

PUSH DX

; Обработка строки

CALL ProcessString

MOV DX, OFFSET Recovery\_9h

CALL WriteMSG

MOV BYTE PTR [Recovery\_9h], '$' ; Устанавливаем в первый символ строки знак конца строки

; Восстановление 9h

PUSH DS ; Сохраняем текущий сегмент данных (DS) на стеке

MOV DX, KEEP\_OLD\_9H ; Смещение для оргинального обработчика 9h

MOV AX, KEEP\_OLD\_9H+2 ; Сегмент для оргинального обработчика 9h

MOV DS, AX ; Загружаем сегмент обработчика 9h в регистр DS

MOV AH, 25h ; Функция 25h прерывания 21h для замены вектора прерывания

MOV AL, 9h ; Указываем, что восстанавливаем обработчик для прерывания 9h

INT 21h ; Выполняем прерывание 21h для восстановления обработчика 9h

POP DS ; Восстанавливаем исходное значение сегмента данных (DS)

POP DX

POP AX

JMP DWORD PTR KEEP\_OLD\_9H ; Переход к старому обработчику

NEW\_INT\_9H ENDP

ProcessString PROC

PUSH AX ; Сохраняем регистры, которые будут изменены

PUSH BX

PUSH CX

PUSH SI

PUSH DI

CLD ; Очищаем флаг направления, чтобы операции с памятью шли по возрастанию адреса

MOV CL, STR1HEAD[1] ; Длина введенной строки

MOV AL, 0 ; Обнуляем AL, это будет использоваться для текущего символа

MOV BX, 0 ; Обнуляем BX, это будет счётчик количества одинаковых последовательных символов

MOV SI, OFFSET STR1 ; Указываем на начало строки STR1

MOV DI, OFFSET STR2 ; Указываем на начало строки STR2

LODSB ; Загружаем первый символ в AL

NEXT\_CHAR:

CMP CL, 0 ; Проверяем, не достигли ли конца строки

JE END\_PROCESS ; Если да, завершаем обработку

MOV AH, AL ; Сохраняем текущий символ

LODSB ; Загружаем следующий символ в AL

DEC CL ; Уменьшаем длину оставшейся строки

CMP AH, AL ; Сравниваем текущий и следующий символ

JNE SKIP ; Если не совпадают, пропускаем

INC BX ; Если символы одинаковы, увеличиваем счётчик (BX)

SKIP:

JMP NEXT\_CHAR ; Переходим к следующему символу

END\_PROCESS:

MOV AX, BX ; Загружаем количество в AX

CALL AddNumberToSTR2 ; Добавляем число в STR2

; Добавляем символ '$' в конец STR2 для завершения строки

MOV AL, '$'

STOSB ; Записываем символ '$' в STR2

MOV DX, OFFSET RESULT ; Указываем на начало строки STR2

CALL WriteMSG ; Выводим на экран

MOV DX, OFFSET STR2 ; Указываем на начало строки STR2

CALL WriteMSG ; Выводим на экран

; Восстанавливаем все регистры, которые были сохранены

POP DI

POP SI

POP CX

POP BX

POP AX

RET

ProcessString ENDP

; Процедура добавления числа в строку STR2

AddNumberToSTR2 PROC

PUSH AX

PUSH BX

PUSH DX

PUSH CX

MOV CX, 0 ; Счётчик цифр (обнуляем)

MOV BX, 10 ; Основание системы счисления (десятичная)

; Разделение числа на цифры

ConvertLoop:

XOR DX, DX ; Очищаем DX перед делением

DIV BX ; Делим AX на 10, остаток в DL, результат в AX

ADD DL, '0' ; Преобразуем остаток в ASCII

PUSH DX ; Сохраняем цифру на стеке

INC CX ; Увеличиваем счётчик цифр

TEST AX, AX ; Проверяем, закончили ли разбиение

JNZ ConvertLoop ; Если AX != 0, продолжаем

; Извлечение цифр из стека в строку

WriteDigits:

POP DX ; Извлекаем цифру из стека

MOV [DI], DL ; Пишем символ в строку

INC DI ; Увеличиваем указатель

LOOP WriteDigits ; Повторяем для всех цифр

POP CX

POP DX

POP BX

POP AX

RET

AddNumberToSTR2 ENDP

END