N 1143
И. А. Полетаева В. М. Стасышин

Программирование на языке ассемблера

ВВЕДЕНИЕ

Заключительная часть методических указаний посвящена вопросам программирования на языке Ассемблера для персональных ЭВМ IBM PC. При разработке программ на Ассемблере можно выделить следующие моменты:

- постановка задачи и составление проекта программы;
- ввод команд программы в ЭВМ с помощью редактора;
- трансляция программы с помощью Ассемблера;
- преобразование результата работы Ассемблера в исполняемый модуль с помощью загрузчика;
- исполнение программы.

Программу для исполнения можно вызвать двумя способами:

- Набрать её имя в качестве команды MS-DOS;
- выполнить программу под управлением отладчика.

Обычно программу следует вызвать из операционной системы DOS только в том случае, если есть уверенность в её безошибочной работе. Пока она не отлажена, вызывайте её под управлением отладчика.

Отладчик DEBUG позволяет изображать и изменять значения переменных ,останавливать исполнение программы в заданной точке или исполнять программу по шагам.

Отладчк AFDPRO предназначен для выполнения программ в пошаговом режиме с целью анализа выполнения программы:

1. Для вызова отладчика запустите модуль afdpro.exe
2. Для выполнения программы под управлением отладчика необходимо ввести команду L имя_программы.exe.
3. С помощью клавиш F1 и F2 осуществляется пошаговое выполение отлаживаемой программы (при нажатии клавиши F2 осуществляется выполнение подпрограмм и циклов, не входя вовнутрь).
4. Результатом выполнения очередной команды является изменение значений регистров процессора, флагов или сегментов прграммы.

   Изменения значений регистров и флагов отображаются в верхнем углу экрана , а изменение значений сегментов программы можно проследить с помощью окон 1 и 2 , которые расположены в правой и нижней частях экрана соответственно.

   Переход из окна в окно осуществляется при использовании клавиш F7, F8, F9, F10.
5. Кроме того, можно изменять значения регистров, флагов и сегментов, отражаемых в окнах 1 и 2. Для этого необходимо перейти в соответствующее окно и произвести необходимые изменения.
6. С помощью команды g сегмент : смещение можно выполнить часть программы до адреса, указанного в качестве аргумента команды g.
7. С помощью клавиши F4 можно получить контекстную подсказку по необходимой команде.
8. Выход из отладчика осуществляется по команде quit.

Лабораторная работа N 1

ОБРАБОТКА СТРОК НА ЯЗЫКЕ АССЕМБЛЕРА

Цель работы

Изучить и приобрести практические навыки работы с командами обработки строк.

Содержание работы

1. Изучить необходимый материал по теме лабораторной работы, руководствуясь методическими указаниями.
2. По предложенному преподавателем варианту разработать программу на языке Ассемблера, решающую поставленную задачу.
3. Отладить программу, убедиться в правильности ее работы на тестовых примерах.

Методические указания

КОМАНДЫ ОБРАБОТКИ СТРОК

Команды обработки строк (КОС) позволяют производить действия над блоками байтов или слов памяти. Эти блоки (или строки) имеют длину до 64 Кбайт.

КОС предоставляют воаможности выполнения пяти основных операций, называемых примитивами, которые обрабатывают строку по одному элементу (байту или слову) за прием. Эти примитивы (пересылка,сравнение/сканирование, загрузка и сохранение) представлены в таблице.Каждый примитив представлен тремя разными командами. В первом формате обработка байт или слов определяется неявно типом операнда (операндов).Второй и третий форматы явно указывают операцию над байтами или словами.

МНЕМОКОД	ФОРМАТ	ФЛАГИ
Префиксы повторения		OF DF IF TF SF ZF AF PF CF
REP REPE / REPZ REPNE / REPNZ	REP REPE REPNE	-- --
Пересылка		
MOVS MOVSB MOVSW	MOVS строка-приемник , строка-источник MOVSB MOVSW	-- --
Сравнение		
CMPS CMPSB CMPSW	CMPS строка-приемник , строка-источник CMPSB CMPSW	+ -- -- -- + + + + + + -- -- -- + + + + + + -- -- -- + + + + +
Сканирование		
SCAS SCASB SCASW	SCAS строка-приемник SCASB SCASW	+ -- -- -- + + + + + + -- -- -- + + + + + + -- -- -- + + + + +
Загрузка и сохранение		
LODS LODSB LODSW STOS STOSB STOSW	LODS строка-приемник LODSB LODSW STOS строка-приемник STOSB STOSW	-- --

-- означает сохранение флага, + - его изменение.

МП 8088 может исполнять только те команды обработки строк, которые не имеют операндов. При трансляции программы Ассемблер всегда преобразует команду с операндами в одну из команд без операндов. МП 8088 предполагает, что строка-приемник находится в дополнительном сегменте, а строка - источник - в сегменте данных. Процессор адресует строку-приемник через регистр DI, а строку-источник - через регистр SI.- Так как команды манипулирования строками предназначены для действий над группой элементов, то они автоматически модифицируют указатели для адресации следующего элемента строки. Бит флага направления DF в регистре флагов определяет, будут значения регистров SI и DI увеличены или уменьшены по завершении команды манипулирования. Если DF=O, то значения регистров SI и DI увеличиваются после исполнения каждой команды; DF=1 - значения уменьшаются.

Состоянием флага DF можно управлять с помощью команды CLD -сбросить флаг направления, которая об-нуляет его, и команды STD -установить флаг направления, которая устанавливает DF=1.

Можно сделать так, чтобы одна команда обработки строк обрабатывала группу последовательных элементов памяти. Для этого перед ней надо указать префикс повторения. Число повторений навлекается из регистра CX.

Пример:

```
MOV  CX, 500                    ;  команда будет повторена 500 раз
REP MOVS DEST, SOURCE
```

Остальные префиксы используют при "решении" о продолжении или прекращении повторении флаг нуля ZF. Следовательно, они приложимы только к командам сравнения строк и поиска значения в строке, которые воздействуют на флаг ZF. Префикс REPE (повторять, пока не равно), имеющий синоним REPZ (повторять, пока не нуль), повторяет команду, пока флаг ZP равен 1 и значение CX не равно 0.

Пример: .

```
MOV CX 100                      ; сравниваются элементы строк до тех пор,
REPNE CMPS   DEST, SOURCE       ;  пока соответствущие элементы не равны; либо пока не
                                                  просматриваются 100 элементов.
```

Обычно регистр SI адресуется к сегменту данных, но с помощью префикса замены сегмента в операнде - источнике можно использовать другой сегмент.

Пример :

```
LEA SI,ES:HERE      ; копируем байт из строки HERE в строку THERE,
LEA DI,ES;THERE     ;  обе  строки  находятся в дополнительном  сегменте.
MOVSB
```

Чтобы скопировать строки в сегменте данных, надо загрузить в регистр дополнительного сегмента ES значение, равное содержимому регистра данных DS, После этого при исполнении команды MOVS МП 8088 будет считать, что копирует (как обычно) строку из сегмента данных в дополнительный сегмент.

Варианты заданий

1. Написать программу формирования сжатой строки символов. Сжатие заключается в удалении пробелов из исходной строки при просмотре ее слева направо.
2. Написать программу формирования сжатой строкн символов. Сжатие заключается в удалении пробелов из исходной строки при просмотре ее справа налево.
3. Написать программу выделения из исходной строки подстроки символов заданной длины с указанного номера позиции.
4. Написать программу, определяющую номер позиции, с которой начинается первое слева вхождение заданной конфигурации символов в исходную строку. .
5. Написать программу формирования строки из исходной путем заданного числа повторений исходной строки.
6. Написать программу, выполняющую следующую фуикцию. Заданы две строки. Проверить вхождение каждого символа строки 1 в строку 2. Если какой-либо (первый слева) символ строки 1 не представлен в строке 2, то фиксируется номер позиции этого символа в строке 1.
7. Написать программу, которая бы инверсировала исходную строку.
8. Написать программу, находящую максимальный и минимальный символы в исходной строке.
9. Написать программу, заменяющую все десятичные цифры в исходной строке на заданный символ.
10. Написать программу, удаляющую из исходной строки повторные вхождения заданного символа.
11. Написать программу, удаляющую пробелы в конце исходной строки.
12. Написать программу, удаляющую из исходной строки заданную конфигурацию символов.

Контрольные вопросы

1. Примитивы.
2. Префиксы повторения.
3. Команды пересылки строк.
4. Замена сегмента (префикс замены сегмента),
5. Команды сравнения строк.
6. Команды сканирования строк.
7. Команды загрузки и сохранения строки.
8. Команды условной передачи управления.

Лабораторная работа N 2

КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ

Цель работы

Изучить команды условной и безусловной передачи управления, ознакомиться с правилами работы с процедурами на языке Ассемблера.

Содержание работы

1. Изучить способы вызова процедуры, возврата из процедуры, вопросы сохранения и восстановления регистров.
2. Написать на языке Ассемблера программу, состоящую из основной части и процедуры, выполняющей определенную в задании функцию.
3. Отладить программу, проверить правильность работы программы на тестовых примерах.

Методические указания

Процедуры

Команды, обеспечивающие исполнение процедур, должны выполнить три функции:

1. Обеспечить сохранение содержимого указателя команд IP. Когда процедура исполнена, .находившийся в этом указателе адрес используется МП 8088 для воэврата к месту вызова.Он называется адресом возврата.
2. .Заставить МП начать исполнение процедуры.
3. Использовать сохраненное содержимое указателя команд IP для возврата в программу и обеспечить продолжение ее исполнения с этого места.

Все эти функции выполняются двумя командами:

CALL - вызвать процедуру ; RET - возвратиться из процедуры.

Команда CALL помещает в стек адрес возврата, занимающий 16 битов, если процедура определена с атрибутом NEAR, и 32 бита, если она определена с атрибутом FAR. Процедуры с атрибутом NEAR могут быть вызваны из того сегмента, в котором они находятся; процедуры с атрибутом FAR могут быть вызваны и из другого сегмента.

Формат команды:

CALL имя

Здесь имя - имя вызываемой процедуры (метка ее начала). Если процедура имя имеет атрибут NEAR, то команда CALL помешает смещение адреса следующей команды в стек. Если процедура имя имеет атрибут FAR, то команда CALL помещает в стек содержимое регистра CS, а затем смещение адреса.

После сохранения адреса возврата команда CALL загружает смещение адреса метки имя в указатель команд IP. Если процедура имеет атрибут FAR, то команда CALL загружает также номер блока метки имя в регистр CS.

Команда RET заставляет МП 8088 возвратиться из процедуры в программу, вызвавшую эту процедуру, делая это "откатом" всего, что сделала команда CALL. Команда RET должна быть последней командой процедуры, исполняемой МП 8088. Это не значит, что команда RET должна стоять в конце процедуры - она лишь исполняется последней.

Вызов процедуры с помощью команды CALL называется прямым вызовом, при котором операндом служит метка начала процедуры. Но можно осуществить и косвенный вызов процедуры через регистр или ячейку памяти. При косвенных вызовах через ячейку памяти МП 8088 извлекает значение указателя команд IP для процедуры из сегмента данных, если не используется регистр BP или не указана замена сегмента. Если для адресации ячейки памяти используется регистр BP, то МП извлечет значение указателя команд IP из сегмента стека.

Можно вызвать процедуру с атрибутом NEAR через регистр:

Пример:

```
CALL  BX
```

В данном случае регистр BX содержит смещение адреса процедуры относительно регистра сегмента CS. При исполнении этой команды МП 8008 копирует содержимое регистра BX в указатель, команд IP, затем передает управление команде, адресуемой парой регистров CS:IP.

Процедуру с атрибутом NEAR можно вызвать косвенно, используя переменную размером в слово:

Пример:

```
CALL   WORD  PTR[BX]
CALL   WORD  PTR[BX][SI]
CALL   WORD  PTR  VARIABLE_NAME
CALL   WORD  PTR  VARIABLE_NAME[BX]
CALL   NEW_WORD
CALL   WORD  PTR  ES:[BX][SI]
```

Последняя команда CALL получает адрес процедуры из ячейки дополнительного сегмента (благодаря указанню ES:); остальные комаиды извлекают адреса процедур из ячеек сегмента данных.

Процедуру с атрибутом FAR можно вызвать косвенно, используя переменную размером в двойное слово:

Пример:

```
CALL   DWORD  PTR[BX]
CALL   MEM_DWORD
CALL   DWORD  PTR  SS:VARIABLE_NAME[SI]
```

Здесь первые две команды CALL извлекают адреса процедур из сегмента данных, последняя - из сегмента стека.

Процедура может сама вызвать другие процедуры. Вызов одной процедуры из другой называется вложением процедур. Так как каждая команда CALL помещает в стек два иди четыре байта адреса, то число уровней вложения ограничено только размером сегмента стека. Поскольку сегмент стека может иметь до 64 Кбайт, то возможности вложения практически не ограничены.

Команда безусловного перехода JMP .

Команда безусловного перехода JMP заставляет МП 8088 извлечь новую команду не из следующей ячейки памяти.

JMP имя

Здесь операнд имя подчиняется тем же правилам, что и операнд команды CALL: ои может иметь атрибут NEAR или FAR , быть прямым или косвенным. При прямом переходе команда JMP занимает три байта, если метка имеет имеет атрибут NEAR, и пять байтов - если атрибут FAR.

Команды условной передачи управления (КУПУ)

У МП 8088 есть 17 различных команд, которые позволяют ему "принять решение" о ходе исполнения программы в зависимости от определенных; условий, например, нулевого значения регистра или единичного значения флага переноса CF. Действие этих команд зависит от результата использования предшествующей команды сравнения CMP или команд вычитания (SUB или SBB).

Формат:

Jx близкая_метка

Здесь x - модификатор, состоящий, из одной, двух или трех букв. Запись операнда , близкая_метка подчеркивает, что метка перехода должна находиться не далее -128 или +127 байтов от команды условной передачи управления.

Сравните: JMP могут передать управление в любое место памяти. КУПУ занимают два байта: первый байт содержит код операции, второй - относительный сдвиг. Использование этих команд занимает 16 тактов, если происходит переход, и 4 такта, если перехода нет. Поэтому при составлении программы старайтесь подбирать такие КУПУ, при которых переход менее вероятен.

Примеры:

1.
```
ADD   AL,BL     ; если при сложении возник перенос,
 JC    TOOBIG     ; осуществляется переход к метке TOOBIG
```

2.
```
SUB   AL.BL     ; если при вычитании в AL результат = O,
JZ    ZERO        ; то переход к ZERO
```

3.
```
CMP   AL,BL     ; если значение регистров AL и BL одинаково
JE   ZERO        ; переход к ZERO.
```

4.
```
CMP   BX,AX     ; переход осуществляется, если содержимое
JA   BxMODE     ; BX > содержимого .AX и операнды не имеют ;знаков
```

```
CMP.  BX,AX     ; переход  осуществляется,  если операнды
JG    BxMODE    ; имеют знак и содержимое BX > содержимого AX
```

КУПУ могут предшествовать любые команды, изменяющие состояния флагов, но обычно они используются совместно с командами сравнения CMP.

Условие перехода	Следующая за CMP команда	
	для чисел без знака	для чисел со знаком
приемник > источник	JA	JG
приемник = источник	JE	JE
приемник ≠ источник	JNE	JNE
приемник < источник	JB	JL
приемник <= источник	JBE	JLE
приемник >= источник	JAE	JGE

Варианты заданий

1. Найти всех соседей заданного символа в исходной строке.Первый и последний символ считать соседями.
2. Подсчитать количество символов, у которых равные соседи и исходной строке. Первый и последний символы считать соседями.
3. Переставить в обратном порядке все символы между первым и последним вхождением заданного символа в исходной строке, если заданный символ встречается в строке не менее двух раз.
4. В исходную строку вставить после заданного символ-а все символы, предшествующие ему. Оставшуюся часть строки оставить без изменения.
5. В исходную строку вставить после заданного символа все символы, предшествующие заданному в обратном порядке. Оставшуюся часть строки оставить без изменения.
6. В последней строке символы, следующие за заданиым символом, переписать в обратном порядке.
7. Образовать строку, повторив фрагмент исходной строки с заданной позиции данной длины требуемое число раз.
8. Образовать строку из исходной, повторив i и элемент 1 раз, 1+1-й элемент 1+1 рае, 1+2 и элемент - 1.2 раза.
9. В исходной строке фрагмент с заданной позиции заданной длины повторить требуемое число раз. Остапьные символы строки оставить без изменения.
10. Часть строки, следующую за первым вхождением заданного символа переписать в обратном порядке заданное число раз.
11. Часть строки, предшествующую первому вхождению заданного символа, переписать в обратном порядке заданное число раз.
12. В исходной строке указанное число символов, начиная с заданной позиции, переписать в конец строки.

Контрольные вопросы

1. Команда вызова процедуры CALL.
2. Команда возврата из процедуры.
3. Прямой и косвенный вызов процедуры.
4. Сохранение и восстановление регистров.
5. Команда безусловного перехода JMP.
6. Команды условной передачи управления.

Лабораторная работа N 3

ПРЕРЫВАНИЕ ОПЕРАЦИОННОЙ СИСТЕМЫ DOS

Цель работы

Изучить прерывание типа 21 (вызов функций) и возможности его применения для ввода и вывода информации в программе пользователя.

Содержание работы

1. Изучить возможности взаимодействия с клавиатурой, дисплеем, принтером и диском, представляемые прерыванием DOS типа 21,
2. Написать программу, реализующую задание с использованием определенной функции, имитируемой прерыванием типа 21.
3. Отладить программу, убедиться в правильности ее работы на тестовых примерах.

Методические указания

Фирма IBM резервирует прерывания типов 20-3F для использования операционной системы DOS. Большинство из этих прерываний полезны только для DOS. Однако прерывание типа 21 (вызов функции) предоставляет пользователю множество удобных возможностей взаимодействия с клавиатурой , дисплеем , принтером, диском и асинхронным последовательным устройством.В данной работе изучаются функции для работы с клавиатурой и дисплеем , инициируемые прерыванием типа 21, перечисленные в таблице.

Вызовы функций, инициируемые прерыванием типа 21

Регистр АН	Операция	Дополнительные регистры	Выходные регистры
	Функции для работы с клавиатурой		
1	Ожидание набора символа на клавиатуре и последующее изображение его на экране (с проверкой на Ctrl-Break) (*)	Не используются	(AL) = символ
6	Чтение символа с клаватуры (без проверки на Ctrl-Break) (*)	(DL)=0FFH	(AL) = очередной символ символ,если буфер клавиатуры не пуст (AL)=0 ,если буфер пуст
7	Ожидание набора символа на клавиатуре без последующего его изображения (без проверки на Ctrl-Break)	Не используются	(AL) = символ
8	То же, что функция 7, но с проверкой на Ctrl-Break	Не используются	(AL) = символ
A	Чтение клавиатурной строки в буфер	(DS:DX) = адрес буфера Первый байт = размер буфера	Второй байт буфера = число фактически прочитанных символов

Продолжение таблицы

Регистр АН	Операция	Дополнительные регистры	Выходные регистры
B	Чтение состояния клавиатуры	Не используются	AL = 0FFH, если клавиатура пуста; AL = 0, если она содержит хотя бы один символ
C	Опустошение буфера клавивтуры и вызов функции для работы с клавиатурой	(AL)=номер функции	В соответствии с вызываемой функцией
	Функции для работы с дисплеем		
2	Изображение символа (с проверкой на Ctrl-Break)	(DL)=символ	Не используются
5	Печать символа (без проверки на Ctrl-Break)	(DL)=символ	Не используются
6	Изображение символа (без проверки на Ctrl-Break)	(DL)=символ	Не используются
9	Изображение строки	(DS:DX) = адрес строки, которая должна оканчиваться знаком $	Не используются

(*) Некоторые комбинации клавиш генерируют "расширенные коды", и для их чтения может потребоваться два вызова функций.

Чтобы после выдачи строки символов курсор перешел на начало следующей строки экрана (в случае, если выдается сообщение, а не приглашение к вводу), надо перед знаком $ вставить символы возврата каретки ODH и перехода на следующую строку OAH, например,

```
MESSAGE   DB  ‘Операция сортировки завершена’
          DB   OOH, OAH, '$'
```

Варианты заданий

Напишите диалоговую программу с использованием функций для работы с клавиатурой и функций для работы с дисплеем, выполняющую задание указанного варианта.

1. Подсчитать количество вхождений заданного символа в строку текста.
2. Заменить заданный символ в строке текста на указанный новый символ.
3. Удалить заданный символ из текста.
4. Подсчитать количество слов в строке, считая словом последовательность знаков между пробелами.
5. Переместить заданный символ, если он содержится в строке, в начало строки.
6. Переместить заданный символ, если он находится в строке, в конец строки.
7. Удалить все пробелы из строки символов.
8. Задана строка слов. Словом считается последовательность символов, разделенная пробелами:

а) в исходной строке оставить между словами лишь .по одному пробелу, удалив лишние;
б) в исходной строке изменить порядок следования слов на инверсный;
в) из исходной строки удалить слова, начинающиеся с заданного символа;
г) в исходной строке слова, начинающиеся с заданной буквы, заменить знаком $;
д) из исходной строки удалить слова, содержащие хотя бы одну десятичную цифру.

Контрольные вопросы

1. Прерывание.
2. Вектор прерывания.
3. Прерывание типа 21.
4. Функции для работы с клавиатурой.
5. Функции для работы с дисплеем.

Лабораторная работа N 4

МАКРООПРЕДЕЛЕНИЯ

Цель работы

Изучить приемы разработки макроопределений использования их в программах.

Содержание работы

1. Изучить состав и средства задания макроопределений.
2. Написать макроопределение, реализующее функцию заданного преподавателем варианта работы (см. описание работы N 3).
3. Написать программу проверки работоспособности разработанного макроопределения.

Методические указания

Каждое макроопределение (МО) имеет три части.

1. Заголовок - псевдооператор MACRO, в поле метки которого указано имя МО, а в поле операнда – необязательный список формальных параметров. В списке формальных параметров указываются переменные - входные параметры, которые могут изменяться при каждом вызове МО.
2. Тело - последовательность операторов Ассемблера (команд и псевдооператоров), которые задают действия, выполняемые МО.
3. Концевик - псевдооператор ENDM, который отмечает конец МО.

Пример :

```
МО для сложения значений размером в слово
ADD  WORDS  MACRO   TERM1,TERM2,SUM
MOV  AX, TERM1
ADD   AX, TERM2
MOV   SUM, AX
ENDM
```

Для Ассемблера безразлично, что будет указано в качестве операндов МО: имена регистров, ячейки памяти или непосредственные значения (конечно, непосредственное значение нельзя использовать в качестве операнда SUM).

Замечание МО легче передать параметры, чем процедуре: в случае МО набирается имя параметра, в случае процедуры необходимо переслать значение параметра в регистр или ячейку памяти.

Псевдооператоры Макроассемблера

ПО общего назначения:

1. MACRO

Формат применения: имя MACRO [список форм.параметров]

. . . . . .

ENDM

Присваивает имя последовательности операторов, которые должны завершиться оператором ENDM.

2. LOCAL

Формат применения: LOCAL [список форм. параметров] заставляет . Ассемблер создать уникальное имя для каждой метки из списка формальных параметров и подставить это имя при каждом вхождении метки в расширение МО.

ПО общего повторения:

1. IRP

формат применения: IRP параметр, <список аргументов>
. . . . . .

SNBM

заставляет Ассемблер повторять находящиеся между IRP и ENDM операторы по одному разу для каждого аргумента списка. При каждом повторении производнтся подстановка очередного аргумента вместо каждого вхождения параметра в блок операторов.

2. IRPC

формат применения: IRPC параметр, строка

. . . . . .

ENDM

заставляет Ассемблер повторять находящиеся между IRPC и ENDM операторы по одному разу для каждого символа строки.

3. REPT

Формат применения: REPT выражение
заставляет Ассемблер повторять находящиеся между REPT и ENDM операторы выражение раз.

Условные ПО

4. EXITM

Формат применения: EXITM
завершает расширение МО в зависимости от результата выполнения условного ПО.

5. IF1 IF1 выражение
Формат применения: . . . . . .
ENDIF

выполняется, если Ассемблер осуществляет первый проход. Обычно используется для включения с помощью оператора INCLUDE файла с библиотекой МО в исходную программу.

6. IFB IFB <аргумент>

формат применения: . . . . . .
ENDIF

выполняется, если <аргумент> пуст. < > обязательны.

7. IFNB IFNB <аргумент>

Формат применения: . . . . . .
ENDIF

выполняется, если <аргумент> не пуст. < > обязательны.

ПО управления листингом:

1. . LALL
Формат применения: . LALL
вызывает выдачу полного листинга (включая комментарии) всех расширений МО. .

2 . SALL.
Формат применения: . SALL
исключает текст МО из листинга.

3. . XALL
Формат применения: . XALL
вызывает печать только тех строк МО, которые генерируют объектный код. Этот режим устанавливается по умолчанию.

Операции в макроопределениях

1. &
Формат: текст & текст
вызывает конкатенацию текста или имен.

2. ;;
Формат: **;;** комментарий
исключает комментарий из листинга, даже если он выдается по команде . LALL.
3. !
формат: ! символ
используется в аргументе для указания Ассемблеру, что символ надо использовать как литерал, а не как имя.

4. %
Формат: % имя
преобразует имя в число. При расширении МО Ассемблер подставляет число вместо имени.

Существуют два способа использования МО: их можно задавать в начале программы или считывать в программу из отдельного файла с библиотекой МО.

Варианты заданий.

Варианты заданий см. в описании лабораторной работы N 3.

Контрольные вопросы

1. Сравнение МО и процедур.
2. Состав макроопределений.
3. Псевдооператоры общего назначения.
4. Псевдооператоры повторения.
5. Условные псевдооператоры.
6. Операции в МО.
7. Задание МО. .

Лабораторная работа N 5

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ НА ЯЗЫКЕ АССЕМБЛЕРА В ПРОГРАММАХ НА ЯЗЫКЕ СИ

Цель работы

Приобрести практические навыки использования в прикладной программе пользователя модулей на языках СИ и Ассемблера.

Содержание работы

1. Изучить условия взаимодействия функций на языке Си с подпрограммой на языке Ассемблера.
2. Изучить правила передачи управления в подпрограмму и обратно.
3. Изучить способы обмена данными между вызывающей функцией и подпрограммой на языке Ассемблера.
4. Написать процедуру на языке Ассемблера, реализующую функцию заданного варианта.
5. Написать вызывающую функцию на языке Си, осуществляющую ввод исходных данных и вывод результатов.
6. Отладить программу, убедиться в правильности ее работы на тестовых примерах.

Методические указания

Взаимодействие языков Си и Ассемблера

Первое требование, которое необходимо выполнить при совместной загрузке обьектных файлов созданных Ассемблером и компилятором языка Си, состоит в том, что все эти объектные файлы должны иметь одинаковый формат, тот, который нужен загрузчику.

Другое требование к взаимодействию состоит в том,что функцию на языке Ассемблера надо кодировать так, чтобы выполнялись соглашения по вызову функций, используемые компилятором языка Си.

Чтобы программа на языке Си (вызывающая функция) могла вызвать подпрограмму на языке Ассемблера, обменяться с ней данными, и соответствующим образом получить управление обратно, должны быть выполне-ны следующие шаги:

1. Программа должна сохранить адрес команды, с которой будет продолжено ее исполнение после завершения вызова подпрограммы. Затем программа передает управление подпрограмме. Программе и подпрограмме нужно придерживаться таких соглашений об испольаовании машинных регистров, чтобы подпрограмма не уничтожала регистровых значений программы (иногда называемых средой).
2. После завершения подпрограмме необходимо возвратить управление по адресу, сохраненному программой.
3. Программа и подпрограмма должны придерживаться общих соглашений о передаче даипых из программы в подпрограмму.
4. Должны быть установлены также соглашения о возвращении данных из подпрограммы в программу.

Обычно для этого используется стек. Микропроцессор 8088 фирмы INTEL имеет отдельный регистр сегмента, называемый регистр сегмента стека SS.В дополнеине к нему имеется регистр управления стека SP. При исполнении опредеденных команд этот регистр обеспечивает использование в качестве стека той области памяти, которая адресуется с помощыо регистра SS.

Компилятор языка Си генерирует соответствующий код, обеспечивающий резервирование памяти для такого стека и присваивание адреса этой области регистрам SS и SP, в качестве начальных значений. Этот стек будем называть стеком времени исполнения.На IBM PC регистру SP в качестве начального значения присваивается адрес последней ячейки сегмента стека, содержимое регистра SP убывает при помещении объектов в стек.

Передача данных подпрограмме

Существуют два способа обмена данными между вызывающей функцией и подпрограммой на языке Ассемблера:

- использование глобальных данных;
- использоваиие аргументов.

Использование глобальных данных

Подпрограмма на языке Ассемблера может иметь доступ к глобальным данным, определенным в модуле на языке Си с использованием внешнего класса хранения данных. В подпрограмме, необходимо определить сегмент данных и использовать для ссылки на внешние объекты псевдооператор EXTERN.

Пример 1.

Подпрограмма изменяет значение внешнего объекта VAL внешние объекты записываются прописными буквами.

```
/*Содержание модуля TEST1.C*/
extern  second ( )     /*объявить внешнюю подпрограмму*/
int  val               /*определить внешний объект данных*/
main ( )
< second ( );
printf ("значение VAL = % d\n",val);
}
;Содержание модуля SECOND. ASM
DGROUP GROUP DATA                       ; сегмент данных подпрограммы
DATA SEGMENT WORD PUBLIC DATA           ; должен быть
        ASSUME DS:DGROUP                ; в том же сегменте,  что и данные программы на языке Си

        ; Псевдооператор GROUP служит для указания Ассемблеру на то, что этот сегмент надо поместить
        ; вместе в один сегмент с данными программы на Си.

        EXTERN VAL:WORD                 ;  объявить  VAL как глобальный объект целого типа DATA ENDS

        PGROUP GROUP PROG               ; сегмент команд надо поместить вместе
                                        ; с командами других модулей в один блок памяти размером 64 К
        PR0G SEGMENT
        BYTE PUBLIC 'PROG'
              ASSUME CS: PGROUP
              PUBLIC SECOND             ; сделать это имя глобальным
        SECOND PROC NEAR                ; малая модель распределений памяти
        MOV VAL,4                       ; модифицировать глобальную переменную
        RET                         ; вернуться в вызывающую программу
        SECOND ENDP                     ; конец программы
        PROG ENDS                       ; конец сегмента
        END                             ; конец файла
```

Чтобы тестировать эту программу, надо откомпилировать модуль на языке Си, откомпилировать подпрограмму с помощью Ассемблера, загрузить исполняемую программу и вызвать ее.

Использование аргументов функции

В языке Си аргументы передаются по значению, т.е. вызываемая подпрограмма или функция получает копию каждого аргумента. Значения аргументов копируются в стек времени исполнения .

Пример 2.

a1, a2, a3 - целые значения.

Вызов third (al, a2, a3) преобразуется компилятором Си в последовательность команд МП 8088:

```
PUSH a3
PUSH a2
PUSH a1
CALL THIRD
```

Замечания

1. Это общепринятый подход, зависящий от реализации компилятора.
2. Аргументы помещаются в стек в порядке. обратном тому, в котором они указаны при вызове функции. Ячейка с адресом возврата имеет адрес [SP]. Копии аргументов имеют адреса [SP]+2, [SP]+4, [SP]+6.

МП 8088 не позволяет использовать регистр указателя стека в команде MOV для извлечения значений, находящихся по этим адресам. В ней следует использовать другой регистр указателя. Обычно для этих целей служит регистр BP (регистр указателя базы). Значение этого регистра часто называют указателем фрейма. Поэтому одной из первых команд любой подпрограммы, которой необходимо адресоваться к аргументам, является помещение в регистр BP значения (SP). Перед этим необходимо сохранить (BP) в стеке, а затем воспользоваться регистром BP в подпрограмме, непосредственно перед возвратом в вызывающую функцию нужно восстановить исходное значение BP.

Пример 3.

Рассмотрим подпрограмму, которая получает три целых аргумента, суммирует их, а затем возвращает полученный результат через внешнюю переменную VAL:

Организация доступа к аргументам функции:

```
/* Содержание модуля TEST2.C */
        extern sum( );
        int val;
        main ( )
        {   int x = 10 ;
            sum  (x, 20, 20+5) ;
            printf ("успешное возвращение из подпрограммы  val = %d\n", val) ;
        }
```

1. Аргументами вызова могут быть идентификаторы, константа или выражения. Каждое выражение вычисляется, а его значение помещается в стек командой PUSH. Однако порядок вычисления таких выражений не оговорен в Си.
2. В языке Си можно передавать такие объекты, как слово, символы, длинные целые значения, значения с плавающей точкой одинарной и двойной точности,а также (если это обеспечивается компилятором) целые структуры.

В этом случае размер части стека, занимаемой каждым аргументом, зависит от его размера. Однако при передаче аргументов компиляторы Си выполняют некоторые упрощения:

1) объекты типов char, short, int занимают в стеке по одному слову;
2) объекты типа long занимают по два слова;
3) объекты типа float, double передаются в формате объектов типа double и требуют в стеке по 4 слова;
4) если компилятор обеспечивает передачу структур по значению, то в стеке требуется столько байтов, чтобы в них поместились все элементы структуры. При этом число выделяемых байтов округляется до целого, кратного размеру слова;
5) для указателя требуется одно или два слова в зависимости от используемой модели распределения памяти.

Наряду с возвращением значений через глобальные объекты подпрограмма на языке Ассемблера может возвращать их через аргументы вызова или как значение подпрограммы.

Чтобы вызывающая программа могла воспринять возвращаемую сумму как значение имени подпрограммы, нужно следующим образом модифицировать модуль TEST2.C (пример 3)

```
/* содержание модуля TEST3.C*/
extern sum ( );
main ( )
{ inl val, x = 10;
      val = sum (x, 20, 20+5) ;
      printf (" Успешное возвращение из подпрограммы , val =% d\n , val) ;
}
```

Вызывающая функция и подпрограмма должны придерживаться определенных соглашений относительно возвращения значения через имя подпрограммы. Многие компиляторы языка Си рассчитаны на то, что это значение запомнено в одном или нескольких регистрах.

Общепринятые соглашения по использоваию этих ресурсов на IBM PC показаны ниже в таблице. Чтобы в соответствии с этими соглашениями модифицировать подпрограмму sum, из программы примера 3 надо удалить ссылки на внешний объект VAL м оставить вычисленный ею результат в регистре AX.

. Замечание. Результат остался в AX, программа на Си, получив управление, извлечет этот результат из AX.

Регистры, используемые для возвращения значений

Тип возвращаемого значения	Используемые регистры
целый (16 битов) длинный целый (38 бита) вещественный с двойной точностью (64 бита)	AX AX, BX AX, BX, CX, DX

Примечание. Это соглашение может быть своим у каждого компилятора .

Использование аргументов

Если подпрограмма должна возвращать несколько значений, то она может использовать для этой цели свои аргументы. В этом случае передается адрес аргумента, т.е. передается аргумент по адресу.

Подпрограмму sum модифицируем: теперь у нее четыре аргумента и результат будет возвращаться через последний из них. Теперь её вызов имеет следущий вид:

```
sum (x, 20, 20+5, & val);
```

Варианты заданий

См. варианта заданий к лабораторной работе 2.

Контрольные вопросы

1. Взаимодействие языков Си и Ассемблера.
2. Передача управления в подпрограмму и обратно.
3. Передача данных подпрограмме.
4. Возвращение значений.

Лабораторная работа М 6

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ НА ЯЗЫКЕ СИ В ПРОЦЕДУРАХ НА ЯЗЫКЕ АССЕМБЛЕА

ВЫЗОВ ФУНКЦИЙ ОПЕРАЦИОННОЙ СИСТЕМЫ DOS ИЗ ПРОГРАММЫ НА ЯЗЫКЕ СИ

Цель работы

Приобрести практические навыки использования в прикладной программе пользователя модулей на языках Ассемблера и Си с вызовом функций операционной системы DOS.

Содержание работы

1. Изучить условия взаимодействия вызывающей подпрограммы на языке Ассемблера с вызываемой функцией на языке Си.
2. На языке Си разработать функцию, решающую задачу одного из вариантов лабораторной работы.
3. Оценить быстродействие реализованного алгоритма сортировки или поиска, использовав для этого соответствующую функцию прерывания типа 21.
4. Функция сортировки или поиска должна реализовать алгоритм для наиболее общего случая применения .В функцию необходимо включить параметры, оценивающие алгоритм, такие, как число просмотров, число перестановок или перемещений и т.д.
5. В основной программе необходимо распечатать исходные данные и результаты задачи (отсортированные данные, число просмотров, перестановок, время поиска, длительность сортировки и т.д.).

Методические указания

Вызов функций на языке Си из программ на языке Ассемблера

Чаще всего приходится вызывать подпрограммы на языке Ассемблера из функций на Си. Однако можно вызывать функции на языке Си из программ на языке Ассемблера. Для этого надо объявить имя функции на языке Си в программе на языке Ассемблера, используя псевдооператор EXTERN, который должен следовать за псевдооператором SESMENT.

Если имя функции на языке Си – myfunc, то для моделей распределения памяти с малыми кодами ее объявление, будет иметь вид

```
EXTERN MYFUNCT:NEAR
```

Чтобы вызвать функцию на языке Си, её каждый аргумент нужно поместить в стек, начиная с последнего аргумента, а затем вызвать функцию с помощью команды CALL. После возвращения из функции на языке Си программа на языке Ассемблера должна очистить стек, удалив из него все ранее помещенные аргументы. Для этого можно с помощью команды POP извлечь их один за другим. Но легче всего просто увеличить содержимое указателя стека на целое значение, .равное числу байтов, ранее помещенных в стек.

Пример 1.

Вызов функции на языке Си из программы на языке Ассемблера.

```
PGROUP  GROUP  PROG
        SEQMENT  BYTE  PUBLIC  'PROG'
        EXTERN MYFUNCT: NEAR            ; имя функции на Си
        ASSUME CS:PGROUP
        PUBLIC ASMFUNCT
ASMFUNCT PROC NEAR
. . . . . .
PUSH AX          ; Предполагается, что значения аргументов
PUSH  BX         ;  находятся в регистрах AX,  BX,  CX
PUSH CX
CALL MYFUNCT     ;  Вызвать функцию
ADD SP,6         ; Очистить стек от значений аргументов
. . . . . .
```

Если данные определены внутри программы на языке Ассемблера, то их можно сделать доступными функциям на языке Си, объявив их общедоступными с помощью псевдооператора PUBLIC в сегменте данных программы на языке Ассемблера и внешними с помощью служебного слова extern в функции на языке Си .

Использование локальных данных

Функции на языке Си используют стек времени исполнения и для других целей: они выделяют в нем память для автоматических и рабочих объектов данных. Это делается путем уменьшения содержимого указателя стека на число байтов, занимаемых объектом данного типа. Например, если в функции определены три целых объекта с классом хранения automatic, то после команды сохранения значения указателя базы при входе в функцию компилятор языка Си должен генерировать команду : SUB SP,6 . Эта команда выделит в стеке для последующего использования весть байтов, доступ к которым можно организовать путем относительной адресации по регистру указателя базы. Если компилятор инициализирует регистр указателя базы до выделения памяти В стеке, то смещения адресов автоматических переменных по отношению к значению регистра указателя базы будет отрицательным. Перед возвращением в вызывающую программу надо очистить стек, восстановив содержимое указателя стека.

Пример 2.

Выделение и освобождение памяти в стеке.

Вариант 1

```
PUSH BP       ; сохранить (BP) при вызове
MOV BP.SP     ; установить новое значение BP
SUB SP.n      ; выделить n байтов
. . . . . .
MOV SP.BP     ; освободить память
POP BP        ; восстановить BP
RET
```

Вариант 2

```
PUSH  BP      ;  сохранить (BP) при вызове
SUB SP,n      ; выделить n байтов
MOV BP,SP     ;  установить новое  (BP)
. . . . . .
ADD SP,n      ; освободить память
POP BP        ; восстановить BP
RET
```

Этими же методами выделения рабочей памяти можно воспользоваться и в подпрограммах на языке Ассемблера.

Замечание. В первом варианте указатель базы устанавливается до выделения в стеке памяти для локальных данных, а во втором -после выделения памяти. В первом варианте указатель базы продолжает показывать на ту ячейку стека, на которую показывал указатель стека до выделения пакета для локальных данных. Поэтому перед выполнением команды RET стек можно очистить от этих данных, присвоив указателю стека значение указателя базы.

Вызов функций операционной системы DOS из программы на яэыке Си

В операционной системе DOS предусмотрено много служебных функций, которыми можно воспользоваться в прикладной программе. Некоторые из них обеспечивают операции обмена данными с клавиатурой и экраном, диском и другими периферийными устройствами. Чтобы воспользоваться такой функцией, необходимо вызвать прерывание микропроцессора 8088, которое передаст управление операционной системе DOS.Многие компиляторы языка Си не обеспечивают непосредственное кодирование команды прерывания в программе на языке Си. В этом случае для доступа к операционной системе DOS необходимо воспользоваться подпрограммой на языке Ассемблера. Некоторые компиляторы дают возможность указывать команду ASM, позволяющую осуществить прямое кодирование команд языка Ассемблера в программе на языке Си.

Варианты заданий

1. Найти элемент в массиве записей, используя алгоритм:

а) линейного поиска;
б) двоичного поиска.

2. Реализовать алгоритм поиска в строке (Кнута. Мориса, Пратта), оценить его эффективность.
3. Реализовать алгоритм поиска в строке (Воуера,—Мура), оценить его эффективность.
4. Отсортировать элементы в массиве записей, используя алгоритм:

а) сортировки с помощью прямого включения;
б) сортировки с двоичным включением;
в) сортировки с помощью прямого выбора;
г) пузырьковой сортировки;
д) шейкерной сортировки;
е) сортировки Шелла;
ж) сортировки с помощью дерева;
з) быстрой сортировки.

Контрольные вопросы

1. Вызов функций на языке Си из программ на языке Ассемблера.
2. Использование локальных данных.
3. Вызов функций операционной системы DOS из программы на языке Си.
4. Манипулирование структурами данных на языке Ассемблера.
5. Поиск в массиве.
6. Сортировка массивов.

Список литературы

1. Скэндон М Персональные ЭВМ IBM PC и XT. Программирование на языке Ассемблера. - М . : Радио и связь, 1989.
2. Ю-Чжен Ло, Г.Гибсон. Микропроцессоры семейства 8086/8088. – М . : Радио и связь, 1987.
3. Абель П. Язык Ассемблера для IBM PC и программирование М . : Высшая школа, 1992.
4. Трой Д. Програмирование на языке Си для персонального компьютера IBM PC. – М . : Радио и связь, 1991.
5. Кнут Д. Искусство программирования для ЭВМ. М . : Мир, 1978. - Т.3.
6. Вирт Н. Алгоритмы + структуры данных = программы. – М . :Мир, 1983.
7. Вирт Н. Алгоритмы и структуры данных. – М . : Мир, 1989.