# Лабораторная работа №3. Команды передачи управления

**Цель работы**

Изучение команд передачи управления и получение навыка работы с ними.

**Постановка задачи**

* + 1. С помощью процедуры *Random* сформировать случайное целое число в диапазоне от 0 до 65535;
    2. Четное число (четное с точки зрения математики) записать в ЯП (ячейку памяти), начиная с нулевого элемента массива *data*. Нечетное число (нечетное с точки зрения математики) записать в ЯП, начиная с 128 элемента массива *data*. Все числа, которые больше или равны 50000 записать в ЯП, начиная с 256 элемента массива *data*. Все числа, которые меньше 10000 записать в ЯП, начиная с 384 элемента массива *data*;
    3. Повторять п.1-п.2, пока число распределенных данных по любому из признаков не превысит 127.

**Содержание отчета**

Отчет по лабораторной работе должен содержать номер, название, цель лабораторной работы, задание на лабораторную работу, текст программы на языке ассемблера, распечатки формы в активном состоянии с результатами выполнения лабораторной работы в виде дампа памяти и фрагмента программы, содержащей ассемблерную вставку, а также вывод по работе.

Пояснения к выполнению лабораторной работы:

* 1. В самом начале программы выполнить процедуру *Randomize*;
  2. Ассемблерную вставку (если это необходимо) разбить на две части, до применения процедуры *Random* и после ее применения;
  3. Перед применением процедуры *Random* сохранить необходимые РОН в стеке, а после ее применения восстановить их;
  4. Использование процедуры *Random* выглядит следующим образом:

*A := Random(Range*);

где *А* - переменная типа *Word*.

При этом переменная *А* станет равной: *0 ≤ А < Range*.

**Краткие теоретические сведения**

К командам передачи управления относятся:

JMP (Jump) – Прыгнуть, перескочить, скачок;

Jcc (Jump Condition Code) – Прыгнуть по коду условия. Данная расшифровка аббревиатуры команды является традиционной и официальной. Однако для понимания сути команды можно привести другую расшифровку: Jump if the Condition is true or Continue if it is false – прыгнуть, если условие истинно или продолжить, если оно ложно;

LOOP (Loop) – Цикл;

CALL (Call) – Вызов;

RET (Return) – Возврат;

CMP (Compare) – Сравнить;

CMPXCHG (Compare and eXchange) – Сравнить и обменять;

CMPXCHG8B (Compare and eXchange 8 Bytes) – Сравнить и обменять 8 байтов;

**Команда JMP**

Формат команды: JMPОперанд

В качестве операнда может выступать непосредственный адрес (число) или РОН, или ячейка памяти, содержащие адрес кода программы в памяти, с которого должно продолжиться выполнение программы.

Программа, которую выполняет микропроцессор, храниться в оперативной памяти. После компиляции, все команды превращаются в шестнадцатеричные коды, которые последовательно хранятся в памяти. Каждому коду программы соответствует адрес в памяти, так как память микропроцессорных систем можно представить в виде одномерного массива, где каждому элементу массива присвоен номер (адрес). Это прекрасно видно на рисунке 6 в первой его области: первый столбец – адреса ячеек памяти, второй столбец (через пробел от первого) – шестнадцатеричные коды программы, третий столбец – соответствующая кодам из второго столбца мнемоника команд. Команда JMP указывает микропроцессору, начиная с какого адреса следует дальше выполнять команды.

Команда JMP может осуществлять четыре различных типа перехода:

* переход типа *short* (короткий переход) – если адрес перехода находиться в пределах -128…+127 байт от команды, следующей за командой JMP;
* переход типа *near* (ближний переход) – если адрес перехода находиться в том же сегменте памяти, что и команда JMP;
* переход типа *far* (дальний переход) – если адрес перехода находиться в другом сегменте. Дальний переход может выполняться и в тот же самый сегмент при условии, что в сегментной части операнда указано число, совпадающее с текущим значением *CS*;
* переход с переключением задачи – передача управления другой задаче в многозадачной среде.

При выполнении перехода типа *short* и *near* команда JMP фактически преобразовывает значение регистра EIP, изменяя тем самым смещение следующей исполняемой команды относительно начала сегмента кода. Если операнд – регистр или ячейка памяти, то его значение просто копируется в EIP, как если бы это была команда MOV. Если операнд команды JMP – непосредственно указанное число, то это число воспринимается как число со знаком и его значение суммируется с содержимым EIP, приводя к относительному переходу. В ассемблерных программах в качестве операнда обычно указывают имена меток. Программисту сложно каждый раз вычислять конкретное значение адреса перехода и указывать его в команде. Абсолютно логично возложить это на компилятор. На этапе компиляции вычисляются адреса всех меток и полученные значения записываются во все команды, использующие эти метки, в виде относительного смещения.

**Команда Jсс**

Формат команды: Jссметка

В качестве метки может выступать непосредственный адрес для перехода, а так же РОН или ячейка памяти, содержащие адрес для перехода.

Под аббревиатурой Jсс понимается набор команд (см. таблицу 5), осуществляющий условный переход типа short или near. Команды с мнемоникой Jcc НЕТ. Вместо "сс" после J следуют буквы, определяющие условие перехода. Условиями перехода являются состояния флагов, определенных в команде. Если условие выполняется, то управление передается по адресу, указанному в команде. Если условие не выполняется, то микропроцессор продолжает выполнение программы с команды, следующей за командой условного перехода.

Чаще всего, команда из набора Jcc используется сразу после команды CMP. В этом случае условие приобретает формулировку, соответствующую отношению между операндами в команде CMP. В таблице 5 представлены варианты команды Jcc. Например, если операнды CMP были равны, то команда JE, выполненная сразу после CMP, осуществит переход, так как при вычитании двух равных операндов получится нуль, значит, установится флаг нуля ZF, что является условием перехода в команде JE.

Слова «выше» и «ниже» в таблице 5 относятся к сравнению беззнаковых чисел. Слова «больше» и «меньше» относятся к сравнению чисел со знаком.

Таблица 5

|  |  |  |  |
| --- | --- | --- | --- |
| Мнемоника команды | Описание | Перевод описания | Условие перехода |
| JA | Jump if above | Переход, если выше | CF=0 и ZF=0 |
| JAE | Jump if above or equal | Переход, если выше или равно | СF=0 |
| JB | Jump if below | Переход, если ниже | СF=1 |
| JBE | Jump if below or equal | Переход, если ниже или равно | СF=1 или ZF=1 |
| JC | Jump if carry | Переход, если перенос | СF=1 |
| JCXZ | Jump if CX is zero | Переход, если регистр CX=0 | CX=0 |
| JECXZ | Jump if ECX is zero | Переход, если ECX=0 | ECX=0 |
| JE | Jump if equal | Переход, если равно | ZF=1 |
| JG | Jump if greater | Переход, если больше | ZF=0 и SF=OF |
| JGE | Jump if greater or equal | Переход, если больше или равно | SF=OF |
| JL | Jump if less | Переход, если меньше | SF≠OF |
| JLE | Jump if less or equal | Переход, если меньше или равно | ZF=1 или SF≠OF |
| JNA | Jump if not above | Переход, если не выше | CF=1 или ZF=1 |
| JNAE | Jump if not above or equal | Переход, если не выше или равно | СF=1 |
| JNB | Jump if not below | Переход, если не ниже | СF=0 |
| JNBE | Jump if not below or equal | Переход, если не ниже или равно | CF=0 и ZF=0 |
| JNC | Jump if not carry | Переход, если нет переноса | CF=0 |
| JNE | Jump if not equal | Переход, если не равно | ZF=0 |
| JNG | Jump if not greater | Переход, если не больше | ZF=1 или S≠OF |
| JNGE | Jump if not greater or equal | Переход, если не больше или равно | SF≠OF |
| JNL | Jump if not less | Переход, если не меньше | SF=OF |
| JNLE | Jump if not less or equal | Переход, если не меньше или равно | ZF=0 и SF=OF |
| JNO | Jump if not overflow | Переход, если нет переполнения | ОF=0 |
| JNP | Jump if not parity | Переход, если нечетно (не математически) | PF=0 |
| JNS | Jump if not sign | Переход, если результат положителен | SF=0 |
| JNZ | Jump if not zero | Переход, если не нуль | ZF=0 |
| JO | Jump if overflow | Переход, если переполнение | ОF=1 |
| JP | Jump if parity | Переход, если четно (не математически) | PF=1 |
| JS | Jump if sign | Переход, если результат отрицательный | SF=1 |
| JZ | Jump if zero | Переход, если ноль | ZF=1 |

**Группа команд LOOP**

Команды этой группы организуют цикл, используя в качестве счетчика числа итераций регистр **ECX** (другое название регистра ECX: *counter* – счетчик). В регистр **ECX** должно быть предварительно занесено необходимое число итераций. Команды этой группы реализуют классический цикл с уменьшающимся на 1 на каждой итерации счетчиком и постусловием проверки счетчика на нулевое значение. При уменьшении счетчика флаги процессора не меняются!

**Команда LOOP**

Формат команды: **LOOP****метка\_перехода**

Команда LOOP уменьшает значение регистра ECX на 1 и выполняет переход типа *short* на метку (переход не может быть дальше расстояния в  
-128…+127 байт от команды LOOP), если ECX не равен нулю.

**Команды LOOPE и LOOPZ**

Формат команды: **LOOPE****метка\_перехода**

Формат команды: **LOOPZ****метка\_перехода**

Команда LOOPE (LOOPZ) аналогична команде LOOP, только добавляется проверка флага ZF на равенство единице (ZF=1) по логической связке И с условием неравенства ECX нулю. Применяется данная команда в случае, когда нужно досрочно выйти из цикла.

**Команды LOOPNE и LOOPNZ**

Формат команды: **LOOPNE** **метка\_перехода**

Формат команды: **LOOPNZ** **метка\_перехода**

Команда LOOPNE (LOOPNZ) аналогична команде LOOP, только добавляется проверка флага ZF на равенство нулю (ZF=0) по логической связке И с условием неравенства ECX нулю.

Выполнение этих команд не влияет на флаги! Так как команды группы LOOP не влияют на флаги, их удобно использовать вместе с командами, которые в результате своей работы меняют значение флага ZF, например, с командой сравнения CMP или с командой вычитания SUB.

Недостаток всех команд цикла в том, что они реализуют только короткие переходы. Для работы с длинными циклами используются команды условного и безусловного перехода, описанные выше.

**Команда CALL**

Формат команды: **CALL***операнд*

Сохраняет текущий адрес в стеке и передает управление по адресу, указанному в операнде.

Операндом может быть непосредственное значение адреса (или метка), регистр или ячейка памяти, содержащие адрес перехода.

Часто возникает необходимость выполнения специфической операции в разных местах программы. В этом случае в каждом месте выполнения специфической операции надо продублировать последовательность команд, исполняющих эту операцию. Конечно, такое дублирование раздражает и отнимает много времени. Кроме того, оно значительно удлиняет программу.

На самом деле можно избежать дублирования, если определите повторяющуюся последовательность команд как процедуру. Процедура (или, как часто говорят, подпрограмма) представляет собой совокупность команд, которая написана один раз, но может быть исполнена по мере необходимости в любом месте программы.

Процесс передачи управления из основной части программы в процедуру называется вызовом, т.е. процедура вызывается. При вызове процедуры микропроцессор исполняет ее команды, а затем возвращается к тому месту, откуда был сделан вызов.

Возникают два вопроса: как вызвать процедуру и как микропроцессор возвращает управление в нужное место программы? Таким образом, команды, обеспечивающие исполнение процедур, должны выполнять три функции:

1. Обеспечить сохранение содержимого указателя команд EIP. Когда процедура исполнена, находившийся в этом указателе адрес используется микропроцессором для возврата к месту вызова. Будем называть его **адресом возврата**;

2. Заставить микропроцессор начать исполнение процедуры;

3. Использовать сохраненное содержимое указателя команд EIP для возврата в программу и обеспечить продолжение ее исполнения с этого места.

Все эти функции выполняются двумя командами: CALL (call a procedure -вызвать процедуру) и RET (return from procedure - возвратиться из процедуры).

Команда CALL осуществляет функции запоминания адреса возврата и передачи управления процедуре. Она помещает в стек адрес возврата. Объем данных запоминаемого адреса зависит от типа перехода (см. команду JMP).

Если в качестве адреса перехода указано только смещение, считается, что адрес расположен в том же сегменте, что и команда CALL. При этом выполняется ближний вызов процедуры, т.е. в том же сегменте кода. Процессор помещает значение регистра EIP, соответствующее команде следующей за командой CALL, в стек и загружает в EIP значение операнда, осуществляя тем самым передачу управления:

PUSH EIP

EIP=EIP+*смещение\_к\_нужной\_команде*

Если операнд CALL – регистр или переменная, то его значение рассматривается как абсолютное смещение, если операнд – ближняя метка в программе, то ассемблер указывает ее относительное смещение. Чтобы выполнить дальний CALL в реальном режиме процессор помещает в стек значения регистров CS и EIP и осуществляет дальний переход аналогично команде JMP.

**Команда RET**

Формат команды: **RET[***Im16***]**

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

Команда RET извлекает из стека адрес возврата, записанный туда командой CALL, и загружает его в счетчик команд EIP.

Операнд для RET не обязателен, но, если он присутствует, то после считывания адреса возврата из стека будет удалено указанное количество байтов – это нужно, если при вызове процедуры ей через стек передавались параметры.

**Команда CMP**

Формат команды: CMPО1,О2

где О1 – первый операнд. В качестве первого операнда могут использоваться РОН или значение из памяти.

О2 – второй операнд. В качестве второго операнда могут использоваться РОН, число или значение из памяти.

П и И не могут быть одновременно значениями из памяти!!!

Команда CMP сравнивает содержимое первого и второго операндов путем вычитания второго операнда из первого, результат никуда не сохраняется, а устанавливаются лишь флаги. Единственным результатом этой команды является изменение флагов CF,OF,SF,ZF,AF,PF в соответствии с результатом вычитания. В основном эта команда используется вместе с командами условного перехода.

**Команда CMPXCHG**

Формат команды: CMPXCHGО1,О2

Если О1 8-ми разрядный РОН или ячейка памяти, то О2 должен быть 8-ми разрядным РОН. При этом, сравниваются значения регистра AL и О1 путем вычитания из AL значения О1 и, если значения равны, то ZF=1 и О1=О2, иначе ZF=0 и AL=О1.

Если О1 16-ти разрядный РОН или ячейка памяти, то О2 должен быть 16-ти разрядным РОН. При этом, сравниваются значения регистра AX и О1 путем вычитания из AX значения О1 и, если значения равны, то ZF=1 и О1=О2, иначе ZF=0 и AX=О1.

Если О1 32-ух разрядный РОН или ячейка памяти, то О2 должен быть 32-ух разрядным РОН. При этом, сравниваются значения регистра EAX и О1 путем вычитания из EAX значения О1 и, если значения равны, то ZF=1 и О1=О2, иначе ZF=0 и EAX=О1.

Как и в команде CMP, результаты вычитания никуда не сохраняется, а устанавливаются лишь флаги СF,OF,SF,ZF,AF,PF в соответствии с результатом вычитания.

**Команда CMPXCHG8B**

Формат команды: CMPXCHG8BО1

где О1 – первый операнд. В качестве первого операнда может использоваться только 64-х разрядное значение из памяти.

Команда CMPXCHG8B сравнивает содержимое пары регистров EDX:EAX и первого операнда путем вычитания первого операнда из пары регистров EDX:EAX. Результат вычитания никуда не сохраняется. Если значения равны, то ZF=1 и О1=ECX:EBX, иначе ZF=0 и EDX:EAX=О1. Флаги СF,OF,SF,AF,PF устанавливаются в соответствии с результатом вычитания.

**Примерные контрольные вопросы и задачи**

* + 1. Влияет ли результат команды LOOPE на флаги?
    2. Какой регистр выступает в роли счетчика при организации циклов с помощью команды LOOP?
    3. Какой флаг учитывает команда LOOPNZ при переходе к началу цикла?
    4. Какой флаг учитывает команда LOOPE при выходе из цикла?
    5. Чем отличается условный переход от безусловного? Какие команды осуществляют условный переход?
    6. Какой тип переходов (ближний или дальний) реализуются командами LOOP,LOOPE,LOOPNE?
    7. Сохраняет ли команда безусловного перехода JMP информацию о точке возврата?
    8. Какие значения примут регистры EAX, ECX после выполнения следующего фрагмента программы:

MOV EAX,$2

mov ecx, 10   
@loop\_start:   
     add eax, ecx  
     loop @loop\_start