# Лабораторная работа №4

Команды передачи управления

**Цель работы**: изучение команд перехода для организации циклов и ветвлений, получение навыков создания процедур с аргументами.

**Теоретические сведения**

Команды передачи управления включают команды безусловного перехода (JMP), условного перехода (Jxx), организации циклов (LOOPxx), вызова подпрограмм (CALL), возврата из подпрограмм (RET), вызова обработчика прерывания (INT) и возврата из обработчика прерывания (IRET). Передача управления – изменение адреса текущей выполняемой команды (значения содержимого EIP) на то значение, которое закодировано в коде команды как адрес (метка) для перехода, либо как смещение относительно текущего значения EIP (команда LOOPxx). В отличие от команд передачи управления, после выполнения обычных команд процессор просто увеличивают значение EIP на значение, которое равно их длине и считывает следующую команду из сегмента кода

JMP <метка>, JMP <адрес>. Команда безусловного перехода (аналог *goto* в языках высокого уровня). В следующем примере после команды   
JMP j1 выполняется команда MOV ECX, 5:

MOV AX, CX JMP EBX ; Переход по адресу,

JMP j1 записанному в EBX

XOR ECX, ECX

j1: MOV ECX, 5

В системе команд процессора есть набор команд условного перехода, которые передают управление на выполнение команды по заданному адресу или метке в зависимости от установленных флагов. В противном случае выполняется следующая за командой перехода команда. К примеру, если EDI=0, то происходит переход на команду MOV AX, 4:

DEC EDI

JZ j1

XOR EDI, EDI

j1: MOV AX, 4

Очень часто команды безусловного перехода применяются сразу после команды сравнения CMP <операнд\_1>, <операнд\_2>. Данная команда выполняет вычитание также, как и команда SUB, но она только устанавливает флаги и не изменяет значения операндов. В следующей таблице представлен список команд условного перехода, используемых после команды CMP, и соответствующие условия перехода.

| Тип операндов | Мнемокод | Критерий условного  перехода | Значения флагов для перехода |
| --- | --- | --- | --- |
| Любые | JE | операнд\_1 = операнд\_2 | ZF = 1 |
| Любые | JNE | операнд\_1 ≠ операнд\_2 | ZF = 0 |
| Знаковые | JL/JNGE | операнд\_1 < операнд\_2 | SF ≠ OF |
| Знаковые | JLE/JNG | операнд\_1 ≤ операнд\_2 | SF ≠ OF или ZF = 1 |
| Знаковые | JG/JNLE | операнд\_1 > операнд\_2 | SF = OF и ZF = 0 |
| Знаковые | JGE/JNL | операнд\_1 ≥ операнд\_2 | SF = OF |
| Беззнаковые | JB/JNAE | операнд\_1 < операнд\_2 | CF = 1 |
| Беззнаковые | JBE/JNA | операнд\_1 ≤ операнд\_2 | CF = 1 или ZF = 1 |
| Беззнаковые | JA/JNBE | операнд\_1 > операнд\_2 | CF = 0 и ZF = 0 |
| Беззнаковые | JAE/JNB | операнд\_1 ≥ операнд\_2 | CF = 0 |

Команды JCXZ/JECXZ передают управление по метке в случае, если значение регистра CX/ECX=0. Данные команды удобно использовать для организации циклов, если регистр ECX используется как счётчик. Команды перехода изменяют последовательность выполнения команд за счёт изменения содержимого регистра EIP.

В общем случае, действие команд условного перехода зависит от состояния флагов, которое устанавливает предшествующая команда. Следующие команды действуют в зависимости от состояния конкретного флага:

|  |  |  |
| --- | --- | --- |
| Название флага | Команда условного перехода | Состояние флага |
| Переноса CF | **JC** | **CF=1** |
| Чётности PF | **JP** | **PF=1** |
| Нуля ZF | **JZ** | **ZF=1** |
| Знака SF | **JS** | **SF=1** |
| Переполнения OF | **JO** | **OF=1** |
| Переноса CF | **JNC** | **CF=0** |
| Чётности PF | **JNP** | **PF=0** |
| Нуля ZF | **JNZ** | **ZF=0** |
| Знака SF | **JNS** | **SF=0** |
| Переполнения OF | **JNO** | **OF=0** |

Флаги размещены в регистре флагов EFLAGS центрального процессора. Младшая половина имеет название FLAGS, имеет размерность 16 бит и содержит следующие флаги:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | NT | IOPL | | OF | DF | IF | TF | SF | ZF | 0 | AF | 0 | PF | 1 | CF |

Установка или сброс флагов зависит от результата выполнения текущей команды процессора. Таким образом, флаги отражают результат последней выполненной команды.

|  |  |  |
| --- | --- | --- |
| Флаг | Условие установки | Пример |
| CF | При заёме или переносе старшего знакового бита. | XOR EAX, EAX  SUB EAX, 1 ; CF = 1 |
| PF | Если младший байт результата содержит чётное число единиц. Данный флаг может быть использован для контроля правильности передачи данных. | XOR EAX, EAX  ADD AX, 111111b ; PF = 1  SUB AX, 1 ; PF = 0 |
| OF | При заёме или переносе в старший знаковый бит. | MOV AL, 01111111b  ADD AL, 1 ; OF = 1 |
| ZF | Если результат выполнения команды – ноль. | MOV EAX, 3  SUB EAX, 3 ; ZF = 1 |
| SF | Дублирует значение старшего (знакового) бита | XOR EAX, EAX  ADD AL, 1 ; SF = 0  SUB AL, 2 ; SF = 1 |
| DF | Изменяется программно командами STD и CLD. Определяет направление обработки массивов в цепочечных командах (STOS, LODS, CMPS и т.д.). | CLD (DF = 0)  STD (DF = 1) |
| AF | При заёме или переносе из 3-го бита в  4-ый. Используется при работе двоично-десятичной арифметикой (BCD-числами) | MOV AX, 1111b  ADD AX, 1 |
| TF | Установка этого флага разрешает пошаговый режим отладки, когда после каждой выполненной инструкции происходит прерывание программы и вызов специального обработчика прерывания. |  |

Для считывания и установки некоторых флагов можно использовать команды LAHF и SAHF. Команда LAHF загружает младший байт регистра флагов в регистр AH={SF,ZF,0,AF,0,PF,1,CF}, а команда SAHF, наоборот, записывает значение из AH в младший байт EFLAGS. Значения некоторых флагов можно устанавливать специальными командами – STC (CF=1), CLC (CF = 0), CLD (DF = 0), STD (DF = 1).

Подобно циклам *for* и *while* на языках высокого уровня на ассемблере можно создавать циклы с помощью следующих команд:

LOOP <метка>

Данная команда сначала выполняет уменьшение на единицу содержимого регистра CX/ECX. Затем она производит сравнение: если CX/ECX>0, то управление передаётся на метку. Если CX/ECX=0, то управление передаётся следующей за LOOP команде.

|  |  |
| --- | --- |
| Пример:  Сложение квадратов чисел от 1 до 10:  102 + 92 + … + 12 | **MOV ECX, 10**  **XOR EBX, EBX**  **j1: MOV EAX, ECX**  **IMUL EAX**  **ADD EBX, EAX**  **LOOP j1** |

LOOPE/LOOPZ <метка>

Команда уменьшает значение CX/ECX на единицу. Если CX/ECX>0 и ZF=1, то управление передаётся на метку перехода. Если CX/ECX=0илиZF=0, то выполняется следующая за LOOP команда.

LOOPNE/LOOPNZ <метка>

Действие команды аналогично предыдущей, но выход из цикла происходит, если CX/ECX=0 или ZF=1. Последние две команды удобно использовать, если необходимо организовать досрочный выход из цикла. Недостатком данных команд LOOPcc является невозможность делать большие переходы, т.е. смещение метки относительно текущего значения EIP не может быть больше 128 байт.

|  |  |
| --- | --- |
| Пример:  Поиск элемента *a* в массиве *mas* с конца. Если элемент найден, происходит выход из цикла | **mas dw 1, 2, 3, 5, 7, 8, 9, 10, -8, 5**  **a dw 7**  **…**  **MOV ECX, 10**  **MOV AX, a**  **j1:**  **CMP mas[ECX\*2-2], AX**  **LOOPNE j1** |

Команды для работы со стеком

PUSH <операнд>. Команда сначала уменьшает значение ESP на 2/4, в зависимости от размера помещаемых данных, а затем записывает в вершину стека по адресу ESP значение операнда размером 2/4 байта.

POP <операнд>. Команда сначала выталкивает 2/4 байта, начинающиеся с адреса ESP, из стека в регистр или ячейку памяти, а затем увеличивает значение регистра ESP на 2/4, в зависимости от размера пересылаемых данных.

PUSHA/PUSHAD. Данная команда не имеет операндов. Она сначала уменьшает значение SP/ESP на 16/32, а затем помещает в стек содержимое регистров общего назначения в следующем порядке: AX/EAX, CX/ECX, DX/EDX, BX/EBX, SP/ESP, BP/EBP, SI/ESI, DI/EDI.

POPA/POPAD. Команда выталкивает из стека значения регистров общего назначения в следующем порядке: DI/EDI, SI/ESI, BP/EBP, SP/ESP, BX/EBX, DX/EDX, CX/ECX, AX/EAX. Содержимое SP/ESP при этом не восстанавливается. Затем значение ESP увеличивается на 16/32.

PUSHF/PUSHFD. Команда сохраняет значение регистра флагов FLAGS/EFLAGS размером 2/4 байта в стеке.

POP/POPFD. Команда восстанавливает значение размером 2/4 байта из вершины стека в регистр флагов FLAGS/EFLAGS.

CALL <имя\_подпрограммы>. Команда предназначена для вызова подпрограммы. Она помещает адрес следующей после CALL команды в стек и передаёт управление по адресу подпрограммы.

RET <операнд>. Осуществляет возврат в вызывающую подпрограмму по адресу, хранящему на вершине стека. Команда извлекает из стека адрес возврата, помещает его в EIP, и увеличивает содержимое ESP на значение, которое имеет непосредственный операнд, выталкивая тем самым из стека данные.

Существуют определённые правила организации подпрограмм и передачи им аргументов. В операционной системе Windows принято вызывать подпрограммы, предварительно передавая их аргументы в стек.

|  |  |
| --- | --- |
| Пример вызова функции с  тремя аргументами | PUSH EAX  PUSH DX  PUSH 100  CALL f |

Допустим изначально ESP=0002FFFC. Тогда после выполнения данного фрагмента кода после входа в подпрограмму, стек будет содержать следующую информацию:

Адрес возврата (EIP)

100

DX

EAX

0002FFFC-0002FFFF

0002FFF2-0002FFF5

0002FFEE-0002FFF1

0002FFF8-0002FFFB

0002FFF6-0002FFF7

ESP=0002FFEE

Адреса ячеек ОП

Содержимое

После выполнения подпрограммы стек должен быть возвращён в то состояние, которое он имел перед выполнением первой команды PUSH EAX. Очистить стек от аргументов обязана вызываемая подпрограмма путём выполнения команды RET 10. 10 байт – суммарный размер аргументов. Если подпрограмма использует для вычислений какие-либо регистры, то необходимо сохранить их содержимое в начале подпрограммы в стеке, а в конце подпрограммы восстановить из стека. Таким образом, для вызывающей подпрограммы значения регистров не изменятся. Для удобства можно задать регистру-указателю кадры стека EBP такое значение, чтобы он указывал на последний помещённый в стек аргумент.

Если подпрограмма является функцией и возвращает какие-либо значения, то их необходимо поместить в регистр AL/AX/EAX/EDX:EAX, если возвращаемое значение имеет размер соответственно 1/2/4/8 байт. Если функция возвращает, к примеру, число типа *long* размером 8 байт, то старшую половину результата нужно поместить в EDX, младшую – в EAX. Остальные регистры, которые не используются для возврата значений из подпрограммы, требуется восстанавливать до исходного состояния, которое они имели в начале подпрограммы.

|  |  |
| --- | --- |
| Пример функции с тремя  аргументами  int *f*(int *x*, int\_16 *y*, int *z*),  вычисляющей значение  выражения *xy* + *z*.  *x*, *z* – знаковые 4-байтовые  аргументы,  *y* – знаковый 2-байтовый.  Аргументы при этом передаются в стек в обратном порядке:  **PUSH EAX ; z (4 байта)**  **PUSH DX ; y (2 байта)**  **PUSH 100 ; x (4 байта)**  **CALL f** | **f proc**  PUSH EBX ; EBX сохранить в стеке  PUSH EBP ; EBP сохранить в стеке  MOV EBP, ESP ; В EBP скопировать ESP  ADD EBP, 12 ; Сместить EBP ближе к аргументам  MOV AX, [EBP+4] ; В AX поместить второй аргумент y  CWDE  MOV EBX, [EBP] ; В EBX поместить первый аргумент x  IMUL EBX  ADD EAX, [EBP+6] ; К EAX прибавить третий аргумент z  POP EBP ; Восстановить EBP  POP EBX ; Восстановить EBX  RET 10 ; Очистка стека от аргументов и возврат в вызывающую подпрограмму  **f endp** ; **Результат – в EAX** |

Задания для выполнения к работе

1. Написать программу для вычисления значения арифметического выражения, используя команды условного и безусловного перехода согласно варианту задания. Подобрать набор тестовых данных   
   (не менее 3). При выполнении операций с числами, преобразовывать их к 4-байтовым числам со знаком. Результат вывести на экран.
2. Написать программу для вычисления значения арифметического выражения, содержащего функцию. Вычисление функции организовать в виде отдельной подпрограммы по всем правилам, описанным выше. Для обработки массивов использовать команды для работы с циклами и команды условного перехода. Подобрать набор тестовых данных (не менее 3). Результат вывести на экран.

Пример выполнения первого задания:

|  |  |  |
| --- | --- | --- |
| # |  | *x* – знаковое однобайтовое,  *y* – беззнаковое двухбайтовое,  *z* – знаковое 4-байтовое |

Программа на ассемблере, вычисляющая данное выражение, имеет вид:

.386

.model flat, stdcall

option casemap: none

include d:\masm32\include\kernel32.inc

include d:\masm32\include\msvcrt.inc

includelib d:\masm32\lib\kernel32.lib

includelib d:\masm32\lib\msvcrt.lib

.data

x db -10

y dw 40

z dd -100

format db "a = %d", 0

.code

start:

XOR ECX, ECX ; Обнулить ECX

MOV CX, y ; Поместить у в младшую половину ECX. y - беззнаковое, поэтому расширять его не нужно. Достаточно обнулить старшие байты. В ECX - 4-байтовое

MOV AL, x ; В AL поместить x

CMP AL, 10 ; Если AL >= 10, то переход на j2

JGE j2

CBW ; Расширение x до 2 байт

CWDE ; Расширение x до 4 байт

CMP AL, 5

JLE j1 ; Если AL <= 5, то переход на j1

IMUL ECX ; EDX:EAX = x\*y

IMUL EAX, 5 ; EAX = 5\*x\*y

JMP j\_out

j1:

ADD EAX, ECX ; EAX = x + y

JMP j\_out

j2:

MOV EAX, z ; EAX = z

IMUL z ; EDX:EAX = z2

ADD EAX, ECX ; EAX = z2 + y

j\_out:

push EAX

push offset format

call crt\_printf ; Вывод результата на экран

add ESP, 8

call crt\_\_getch ; Задержка ввода

push 0

call ExitProcess ; Выход из программы

end start

Набор тестовых данных

|  |  |  |  |
| --- | --- | --- | --- |
| x | y | z | a |
| -10 | 40 | 10 | 30 |
| 7 | 50 | 100 | 30 |
| 100 | 100 | -100 | 10100 |

Пример выполнения второго задания:

|  |  |  |
| --- | --- | --- |
| # |  | *x* – массив 2-байтовых знаковых чисел  *y* – массив однобайтовых знаковых чисел  *h* – двубайтовое знаковое  *n* – беззнаковая переменная размером 2 байта |

Ниже приведён текст программы для вычисления данного выражения:

.386

.model flat, stdcall

option casemap: none

include d:\masm32\include\kernel32.inc

include d:\masm32\include\msvcrt.inc

includelib d:\masm32\lib\kernel32.lib

includelib d:\masm32\lib\msvcrt.lib

.data

x dw -10, -2000, 12, 15, 5, -50, 170, 8, 45, 10

y db 100, -100, 7, -15, 30, 20, 35, 40, 10, 10

h dw -50

n dw 3

format db "a = %d", 0

.code

; Функция от одного аргумента

**f proc**

MOV AL, [ESP+4] ; Аргумент смещён относительно вершины стека на 4 байта

CBW ; Расширение аргумента до 2 байт

CWDE ; До 4 байт

IMUL EAX ; Вычисление квадрата EDX:EAX = y2

ADD EAX, 50 ; EAX = y2 + 50

ret 2 ; Выталкивание из стека 2 байт и возврат в основную программу

**f endp**

start:

XOR EBX, EBX ; В EBX будет накапливаться сумма

XOR ESI, ESI ; ESI - индекс i элементов в массивах

XOR ECX, ECX ; ECX - счётчик итераций

MOV CX, n ; CX = n

j1:

MOV AX, x[ESI\*2] ; AX = xi

CWDE ; Т.к. x - беззнаковое, нужно расширить его до 4 байт

IMUL EAX ; EDX:EAX = xi2

ADD EBX, EAX ; EBX = EBX + xi2

MOV AX, h

CWDE ; Расширение h до размерности двойного слова

MOV EDI, EAX ; Переместить делитель h в EDI

PUSH word ptr y[ESI] ; Поместить в стек yi. Поскольку команда PUSH не может поместить в стек 1 байт, то помещаем 2 байта, т.е. yi и yi+1. Но в процедуре используем только yi

call f ; Вызов функции. Результат: EAX = xi2 + 50

CDQ ; Расширение делимого до 8 байт перед операцией деления

IDIV EDI ; EAX = f(yi) / h. В EDX - остаток от деления

ADD EBX, EAX ; EBX = EBX + f(yi) / h

INC ESI ; i = i + 1

LOOP j1 ; ECX = ECX - 1. Переход в начало цикла, если ECX ≠ 0

MOV EAX, EBX ; Поместить результат в EAX

push EAX

push offset format

call crt\_printf ; Вывод результата на экран

add ESP, 8

call crt\_\_getch ; Задержка ввода

push 0

call ExitProcess ; Выход из программы

end start

Набор тестовых данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x = {-10} | y = {100} | h = -50 | n = 1 | a = -101 |
| x = {-10, 3} | y = {10, 3} | h = 100 | n = 2 | a = 110 |
| x = {20, 1, 1000} | y = {10, 5, 100} | h = -10 | n = 3 | a = 1001428 |

Варианты заданий задачи №1

|  |  |  |
| --- | --- | --- |
| Вари-ант | Выражение | Размерность и тип переменных |
| 1 |  | *x*, *y*, *z* – знаковые числа размером  2 байта |
| 2 |  | *x* – беззнаковое число размером один байт  *y*, *z* – знаковые числа размером 2 байта |
| 3 |  | *x* – знаковое  4-байтовое  *y* – беззнаковое  2-байтовое  *z* – знаковое однобайтовое |
| 4 |  | *x* – беззнаковое однобайтовое  *y*, *z* – знаковое  2-байтовое |
| 5 |  | *x* – беззнаковое  2-байтовое  *y* – знаковое  4-байтовое  *z* – знаковое однобайтовое |
| 6 |  | *x* – знаковое однобайтовое  *y*, *z* – беззнаковое однобайтовое |
| 7 |  | *x* – знаковое однобайтовое  *y*, *t* – беззнаковое  2-байтовое |
| 8 |  | *x* – знаковое однобайтовое  *y* – знаковое  4-байтовое  *z* – беззнаковое  2-байтовое |
| 9 |  | *x, z* – знаковые  2-байтовые числа  *y* – знаковое однобайтовое |
| 10 |  | *x* – знаковое  2-байтовое  *y* – беззнаковое однобайтовое  *z* – беззнаковое  2-байтовое |
| 11 |  | *x* – знаковое  4-байтовое  *y* – знаковое  2-байтовое  *z* – беззнаковое однобайтовое |
| 12 |  | *x*, *z* – знаковое однобайтовое  *y* – знаковое  4-байтовое |
| 13 |  | x – беззнаковое  2-байтовое  y – знаковое однобайтовое  z – беззнаковое однобайтовое |
| 14 |  | x – знаковое однобайтовое  y – знаковое  2-байтовое  z – беззнаковое  2-байтовое |
| 15 |  | *x*, *z* – знаковое  4-байтовое  *y* – знаковое  2-байтовое |
| 16 |  | *x* – беззнаковое однобайтовое  *y*, *z, t* – знаковые  2-байтовые числа. |
| 17 |  | *x*, *y*, *z* – знаковые числа размером  2 байта  *k* – беззнаковое однобайтовое число |
| 18 |  | *x*, *y*, *z* – знаковые числа размером  2 байта |
| 19 |  | *x*, *y*, *z* – знаковые числа размером один байт  *k* – беззнаковое  2-байтовое число |
| 20 |  | *x*, *y*, *z* – знаковые числа размером  2 байта |
| 21 |  | *x* – знаковое  2-байтовое  *y* – знаковое  4-байтовое  *z* – беззнаковое однобайтовое |
| 22 |  | *x* – знаковое  2-байтовое  *y* – знаковое  4-байтовое  *z* – беззнаковое  2-байтовое |
| 23 |  | *x* – беззнаковое  2-байтовое  *y*, *z* – знаковое  4-байтовое |
| 24 |  | *x* – беззнаковое однобайтовое  *y* – знаковое  2-байтовое  *z* – беззнаковое  2-байтовое |
| 25 |  | *x* – знаковое  2-байтовое  *y* – беззнаковое однобайтовое  *z* – знаковое однобайтовое |

Варианты заданий задачи №2

|  |  |  |
| --- | --- | --- |
| 1 |  | *x* – массив 2-байтовых беззнаковых чисел  *y* – массив 4-байтовых знаковых чисел  *n* – беззнаковая переменная размером  4 байта |
| 2 |  | *h* – знаковая переменная размером один байт  *x* – массив 1-байтовых беззнаковых чисел  *y* – массив 2-байтовых беззнаковых чисел  *n* – беззнаковая переменная размером  2 байта |
| 3 |  | *h* – беззнаковая переменная размером  2 байта  *x* – массив беззнаковых  1-байтовых чисел  *y* – массив знаковых  2-байтовых чисел  *n* – беззнаковая переменная размером  4 байта |
| 4 |  | *h* – знаковая переменная размером 4 байта  *x* – массив знаковых  4-байтовых чисел  *y* – массив беззнаковых  2-байтовых чисел  *n* – беззнаковая переменная размером  1 байт |
| 5 |  | *h* – беззнаковая переменная размером  2 байта  *x* – массив 4-байтовых знаковых чисел  *y* – массив 2-байтовых беззнаковых чисел  *n* – беззнаковая переменная размером  2 байта |
| 6 |  | *h* – знаковая переменная размером 2 байта  *x* – массив беззнаковых  1-байтовых чисел  *y* – массив знаковых  2-байтовых чисел  *n* – беззнаковая переменная размером  4 байта |
| 7 |  | *h* – знаковая переменная размером 2 байта  *x* – массив беззнаковых  1-байтовых чисел  *y* – массив 2-байтовых знаковых чисел  *m,n* – беззнаковые переменные размером  1 байт |
| 8 |  | *h* – беззнаковая переменная размером  2 байта  *x* – массив знаковых  2-байтовых чисел  *y* – массив знаковых  4-байтовых чисел  *m, n* – беззнаковые переменные размером  2 байта |
| 9 |  | *h* – знаковая переменная размером 1 байт  *x* – массив 1-байтовых знаковых чисел  *y* – массив беззнаковых  2-байтовых чисел  *m* – беззнаковая переменная размером  2 байта |
| 10 |  | *x* – массив 1-байтовых знаковых чисел;  *y* – массив 2-байтовых беззнаковых чисел;  *m* – беззнаковая переменная размером  4 байта |
| 11 |  | *x*, *y* — массивы 2-байтовых знаковых чисел  *n* – беззнаковая переменная размером 1 байт |
| 12 |  | *x* – массив 2-байтовых знаковых чисел  *y* – массив 2-байтовых беззнаковых чисел  *n* – беззнаковая переменная размером  4 байта |
| 13 |  | *x* – массив 1-байтовых беззнаковых чисел  *y* – массив 2-байтовых знаковых чисел  *k* – беззнаковая переменная размером  2 байта  *n* – беззнаковая переменная размером  1 байт |
| 14 |  | *x* – массив 4-байтовых знаковых чисел  *y* – массив 2-байтовых беззнаковых чисел  *n* – беззнаковая переменная размером  2 байта |
| 15 |  | *h* – двухбайтовое беззнаковое  *x* – массив однобайтовых беззнаковых чисел  *y* – массив двухбайтовых знаковых чисел  *n* – беззнаковая переменная размером  4 байта |
| 16 |  | *h* – однобайтовое знаковое  *x* – массив двухбайтовых знаковых чисел  *y* – массив однобайтовых беззнаковых чисел  *n* – беззнаковая переменная размером  1 байт |
| 17 |  | *h* – двухбайтовое беззнаковое  *x* – массив двухбайтовых знаковых чисел  *y* – массив однобайтовых беззнаковых чисел  *n* – беззнаковая переменная размером  2 байта |
| 18 |  | *h* – двухбайтовое знаковое  *x* – массив двухбайтовых беззнаковых чисел  *y* – массив однобайтовых знаковых чисел  *n* – беззнаковая переменная размером  4 байта |
| 19 |  | *h* – двухбайтовое знаковое  *x* – массив 4-байтовых знаковых чисел  *y* – массив однобайтовых беззнаковых чисел  *m* – беззнаковая переменная размером 1 байт |
| 20 |  | *h* – двухбайтовое знаковое  *x* – массив двухбайтовых беззнаковых чисел  *y* – массив однобайтовых знаковых чисел  *n* – беззнаковая переменная размером  2 байта |
| 21 |  | *t* – массив однобайтовых беззнаковых чисел  *r* – массив двухбайтовых знаковых чисел  *n* – беззнаковая переменная размером  4 байта |
| 22 |  | *x* – массив двухбайтовых знаковых чисел  *y* – массив 4-байтовых знаковых чисел  *n* – беззнаковая переменная размером  1 байт |
| 23 |  | *h* – двухбайтовое знаковое  *x* – массив 2-байтовых знаковых чисел  *y* – массив однобайтовых знаковых чисел  *m,n* – беззнаковые переменные размером  2 байта |
| 24 |  | *h* – однобайтовое беззнаковое  *x* – массив однобайтовых знаковых чисел  *y* – массив 2-байтовых беззнаковых чисел  *n, t* – беззнаковые переменные размером  4 байта |
| 25 |  | *h* – 4-байтовое знаковое  *x* – массив 2-байтовых знаковых чисел  *y* – массив однобайтовых беззнаковых чисел  *m* – беззнаковая переменная размером  1 байт |

Контрольные вопросы

1. Какие существуют способы организации циклов на ассемблере?
2. Как работают команды CALL, RET? Как они изменяют стек и регистры?
3. Как передаются аргументы в подпрограммы на ассемблере?
4. Как обратиться к аргументам в стеке без использования команды POP?
5. Как работают команды условного перехода?
6. Как передать подпрограмме аргумент типа *long*?
7. Куда помещается возвращаемое подпрограммой значение?