Министерство образования и науки Российской Федерации
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики
Кафедра высокопроизводительных вычислений и
параллельного программирования

М.А. Клочков

Программирование на языке ассемблер в ОС Linux

Учебно-методическое пособие

Ижевск
2018

УДК 004.382.2-027.1(07)+519.6(07)
ББК 32.973.202я7+22.193я7
К478

Рекомендовано к изданию
учебно-методическим советом УдГУ

Рецензент: д.ф.-м.н., профессор А.П. Бельтюков

Клочков М.А.
К478 Программирование на языке ассемблер в ОС Linux / М.А. Клочков. Ижевск: Изд-во «Удмуртский университет», 2018. – 42 с.

ISBN 978-5-4312-0573-6

Учебно-методическое пособие состоит из набора примеров исходного кода для решения типовых задач на языке ассемблер, а также содержит задания для организации самостоятельной работы студентов. Отличительной особенностью подачи учебного материала является использование возможностей операционной системы Linux Debian, ассемблера nasm и графической среды разработки программного обеспечения sasm.

Пособие может быть рекомендовано для проведения практических занятий по учебным дисциплинам, связанных с изучением устройства и функционирования компьютеров, например: «Системное программирование», «Архитектура вычислительных систем», «Схемотехника ЭВМ», «Микропроцессорные системы», «Цифровая обработка сигналов».

УДК 004.382.2-027.1(07)+519.6(07)
ББК 32.973.202я7+22.193я7

ISBN 978-5-4312-0573-6

ПРЕДИСЛОВИЕ

В предлагаемом пособии систематизирован опыт преподавания учебных дисциплин, в которых изучается структура вычислительных систем, особенности их функционирования и программирование на низкоуровневых языках.

Знания и навыки, полученные при изучении языка ассемблер, позволяют повышать эффективность работы программных приложений, позволяют лучше понимать структуру и порядок выполнения «сторонних» приложений, владеть навыками управления микропроцессорами.

В методическом аспекте преподавания низкоуровневых языков необходимо отметить непревзойденную гибкость администрирования и эффективность организации многопользовательского режима работы, предоставляемых операционной системой Linux Debian. Единожды обеспечив установку и настройку программных пакетов nasm и sasm, преподаватель получает незаменимый инструмент для организации проведения практических занятий с любым количеством обучающихся, которым для минимальной работы необходим лишь текстовый терминал с поддержкой удаленного доступа по протоколу SSH.

Структура данного пособия имеет традиционную форму подачи учебного материала. Имеется несколько типовых разделов, а именно «Регистры центрального процессора и выполнение арифметических операций», «Организация и выполнение условных переходов», «Циклическая обработка информации», «Хранение и обработка данных в числовых массивах», «Создание процедур и модулей», «Использование сопроцессора». В каждом из них размещен теоретический и практический учебный материал с примерами решения типовых задач.

1. Начальная настройка и авторизация

Предполагается, что авторизация осуществляется с внутренних машин Удмуртского государственного университета. Основное программно-техническое обеспечение располагается на сервере, имеющем ip-адрес **192.168.42.218.** До начала работы на сервере необходимо зарегистрировать учетную запись обучающегося. Таким образом, используется стандартная схема авторизации типа «логин/пароль».

Обычно, учебный процесс построен таким образом, что в большинстве компьютерных классов на клиентские узлы работают под управлением операционных систем семейства MS Windows. Поэтому существует проблема подключения к удаленному серверу под управлением ОС Linux Debian с клиентских узлов. Она решается путем использования, например двух программ – PuTTY, Xmanager Enterprise. Для выполнения типовых задач базового уровня освоения компетенций достаточно возможностей первой программы.

Авторизация с клиентских узлов

PuTTY – свободно распространяемый клиент для различных протоколов удалённого доступа, включая SSH, позволяет подключиться и управлять удаленным узлом (например, сервером). В **PuTTY** реализована только клиентская сторона соединения – сторона отображения, в то время как сама работа выполняется на другой стороне.

Чтобы начать работу с приложением, наведите курсор мыши на ярлык на рабочем столе и дважды щелкните левой клавишей:

Либо данное приложение также можно запустить из командной строки – **putty.exe**, либо воспользоваться инструментом «Поиск файлов и папок», либо скачать дистрибутив по ссылке https://putty.org.ru/download.html.

После того, как приложение начнет работать, откроется окно настройки, в поле **Host Name (or IP address)** напишите **192.168.42.218** и в поле **Saved Session** введите символьное обозначение сервера, например цифры **218**.

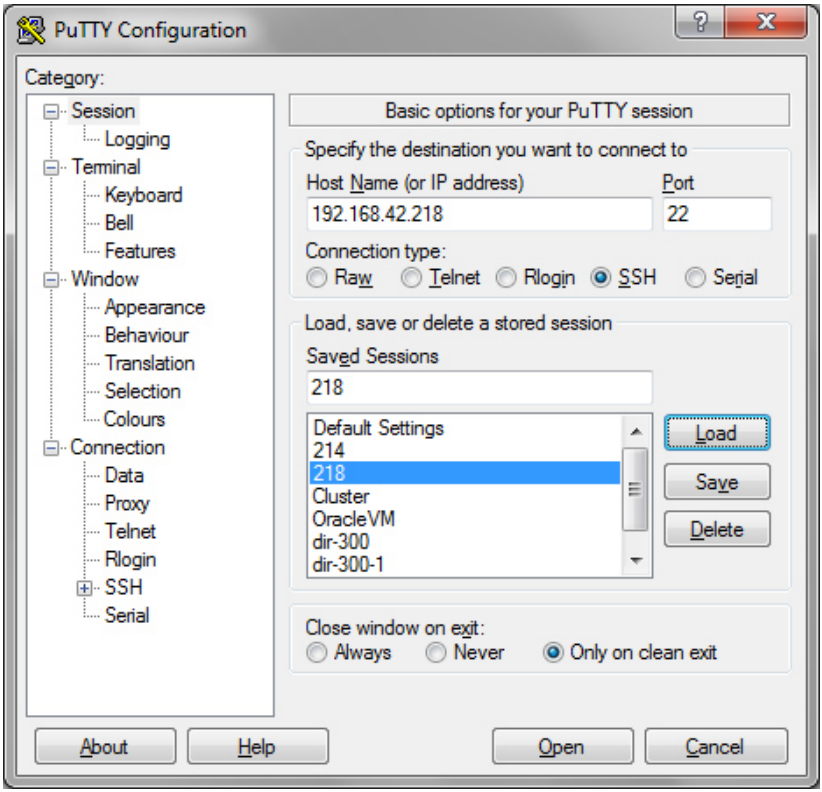

Рис. 1. Настройки соединения **Session.**

В левой части данного окна организован иерархический список настроек. Для соединения с узлом Кластера используются настройки по умолчанию, поэтому имеет смысл указать только некоторые из них, наиболее часто изменяемые пользователями: **Keyboard** и **Translation**.

Часто соединение не устанавливается правильно, если настройки соединения указаны неверно. Внимательно проверяйте ip-адрес сервера.

Для правильного отображения кириллицы на экране терминала необходимо установить в PuTTY тип кодировки UTF-8.

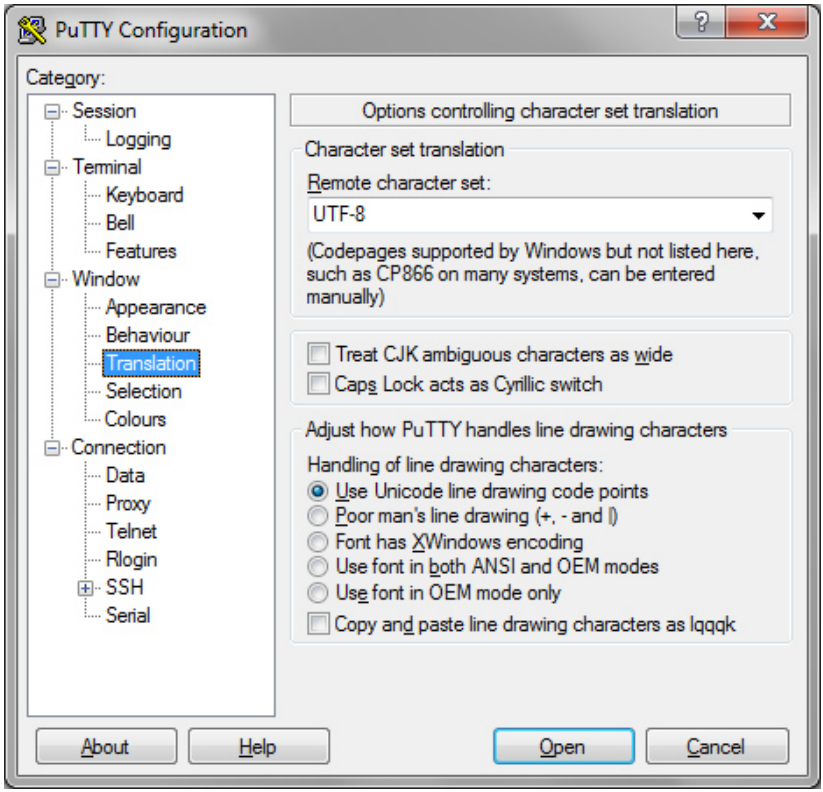

Рис. 2. Настройки соединения **Translation.**

Чтобы сохранить настройки соединения, нажмите кнопку **Save** в окне настроек **Session** (см. Рис. 1). В следующий раз, запуская приложение **PuTTY** для соединения с узлом Кластера, можно будет загрузить настройки соединения. Для этого выберите в списке сохраненных настроек (располагается ниже поля **Saved Sessions) 218** и нажмите кнопку **Load**.

Далее, нажмите кнопку **Open** в левой нижней части окна настроек соединения, и откроется окно для работы на узле как с обычной Linux-системой.

При подключении возможна выдача ошибки, связанной с указанием выбора алгоритма обмена ключами шифрования в протоколе SSH. Для исправления ситуации выберете вкладку как показано на следующем рисунке

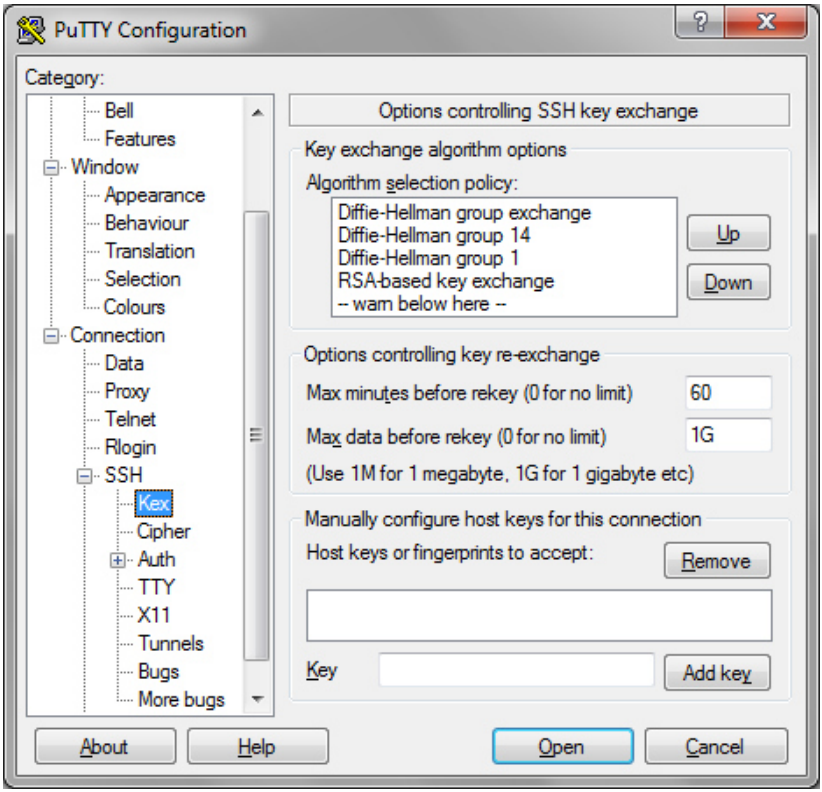

Рис. 3. Настройка выбора алгоритма обмена ключами.

Далее, в окне «Algorithm selection policy» нажимая кнопки «Up» и «Down» необходимо поднять на верхнюю позицию «Diffie-Hellman group 14», и сохранить эти изменения текущей сессии.

Если при подключении возникла ошибка, то проблема скорее всего в сетевых настройках клиентского узла. Необходимо проверить сетевое подключение.

2. Работа в среде Linux Debian

Для дальнейшей работы обучающимся необходима следующая информация по основным командам, используемым в процессе создания программ на языке ассемблер.

Команда *ls* (или *ls -al*) выводит текущий список файлов и каталогов. Данная команда необходима, чтобы найти нужную про-

грамму или исходный код, посмотреть содержимое текущего каталога. После входа в систему по умолчанию это каталог «*/home/учетная запись пользователя*» или содержимое переменной $HOME.

Команда «*cd Имя каталога*» служит для перехода в заданный каталог, «*cd ..*» – переход на один уровень вверх по дереву каталогов. Просто команда *cd* без параметров – возврат в домашний каталог.

Команда «*mkdir Имя каталога*» необходима для создания нового каталога.

Команда «*mv Источник Приемник*» в данном контексте необходима для переименования файла, например «*Источник*» – старое имя файла, «*Приемник*» – новое имя файла.

Команда *nano* используется для запуска простейшего текстового редактора подобного программе «Блокнот» в ОС MS Windows. В данном редакторе «*nano*» предполагается на начальном этапе набирать исходный код.

После его запуска в нижней части окна располагается список доступных команд, обозначенных например «*^G*» Помощь, «*^X*» Выход, где символ «^» обозначает нажатие клавиши «*Ctrl*», то есть для выхода из редактора необходимо нажать комбинацию двух кнопок на клавиатуре «*Ctrl*» + «*X*».

3. Основы работы с ассемблером nasm.

Для того чтобы не усложнять учебный материал на начальном этапе освоения языка ассемблера для написания простейших программ, рекомендуется использовать способ создания приложения из двух частей – части исходного кода на языке *C* для организации ввода/вывода в терминал и второй части исходного кода уже на языке ассемблера. Этот способ сокращает время написания программы и упрощает структуру ассемблерного текста.

Итак, на начальном этапе предлагается использовать утилиту make, которая выполняет команды согласно правилам, указанным в специальном файле. Этот файл называется make-файл (makefile, мейкфайл). Make-файл описывает, каким образом нужно компилировать и компоновать программу.

Предлагается создать следующих два файла, которые будут совсем немного изменяться в будущем.

Первый файл – «*makefile*»

```
runme: main.cpp asm.o
	g++ -static main.cpp asm.o –o runme
asm.o: asm.asm
	nasm –f elf64 asm.asm –o asm.o
```

Данная запись означает, что целью сборки является файл *runme*, который является результатом компиляции двух файлов – «*main.cpp*» (вспомогательный, выполняющий операции ввода/вывода в консоль) и «*asm.asm*» (основной, написанный на ассемблере). Компиляция файла «*asm.asm*» происходит с ключом *–f elf*, что означает использование 32-разрядного формата двоичного файла. Для использования 64-разрядной модели данных необходимо указать ключ *–f elf64*. Важно: при наборе данного текста во второй и четвертой строке сделать вначале отступ, один раз нажав клавишу «TAB» – вставить символ табуляции.

Второй файл – «*main.cpp*»

```
#include <iostream>
using namespace std;
extern "C" int PrintValue() ;
int main(){
  cout << "Результат: "<< PrintValue() << endl ;
  return 0 ;
}
```

Здесь для вывода информации используется внешняя функция *PrintValue()*, описанная в файле «*asm.asm*».

Третий необходимый для данного проекта файл – «*asm.asm*» содержит ассемблерный код. Можно привести следующий простейший пример.

```
global PrintValue
PrintValue:
        mov rax,15
        ret
```

```
----------------------------------------
```

Здесь используется ассемблерная команда *mov* для записи числа *15* в регистр *RAX* – обозначение 64-разрядного регистра общего назначения. Если вы все выполнили верно, то в данном рабочем каталоге появится файл *runme*. Его можно запустить обычным для ОС Linux способом, набрав в командной строке ./runme, то есть поставить точку, слеш и написав имя программы. Комбинация символов «./» означает текущий каталог.

Результатом запуска программы будет текстовое сообщение на экране и цифра *15*.

Разберем следующий пример на вывод информации.

```
----------------------------------------
global PrintValue
section .data
msg db "Привет мир!", 10
msg_len equ $-msg
myvalue dd 15
PrintValue:
        mov rax,4
        mov rbx,1
        mov rcx,msg
        mov rdx,msg_len
        int 80h
        mov rax,[myvalue]
        ret
----------------------------------------
```

Выражение *$-msg* является вычислимым в процессе компиляции, так как знак *$* – псевдометка, обозначающая текущий адрес. В данном фрагменте определены три переменные – *msg* – строка, *msg_len* – длина этой строки и *myvalue* – некоторая числовая переменная типа *dd* «двойное слово».

Обратите внимание, что в исходном коде появилось описание секции *section .data*, что означает раздел описания используемых данных.

Дополнительное задание: в третьей строке после запятой стоит цифра *10*, попробуйте подставить вместо нее любую другую цифру, оцените результат, сделайте выводы.

3. Выполнение арифметических операций

В языке ассемблер имеются следующие команды:

сложение:

```
ADD приемник, источник
INC источник; увеличение источника на единицу
ADC x,y; x := x + y + CF
```

вычитание:

```
SUB приемник, источник
DEC источник; уменьшение источника на единицу
SBB x,y; x := x – y - CF
```

где CF – флаг переноса, то есть операции могут выполняться с учетом значения данного флага

умножение:

```
MUL источник; беззнаковое
IMUL источник; знаковое
```

деление:

```
DIV источник; беззнаковое
IDIV источник; знаковое
```

Пример. Необходимо сложить два числа X=0x00017FF0 и Y=0x00018FFF. Для правильного сложения этих чисел условно разделим каждое из слагаемых на две части, сначала сложим младшие (правые) части, используя команду ADD. Может получиться единица переноса, которую надо учесть при сложении старших (левых) частей. Значит необходимо воспользоваться командой ADC. Положим, что число X размещается в двух 16-разрядных регистрах AX и BX, число Y в CX и DX, тогда исходный код запишется в виде:

```
----------------------------------------
global PrintValue
PrintValue:
        mov rax,0
        mov ax,0x0001
        mov bx,0x7FF0
        mov cx,0x0001
        mov dx,0x8FFF
        add bx,dx
        adc ax,cx
        ret
----------------------------------------
```

Замечание. Результатом будет являться сумма старших разрядов с учетом переполнения сложения младших. Это связано с выводом значения регистра RAX по умолчанию.

Дополнительное задание: удалите команду mov rax,0, как это сказалось на результате? Как изменится результат, если использовать в программе 64-разрядные регистры вместо 16-разрядных? Напишите аналогичную программу для операции вычитания.

Рассмотрим более подробно реализацию операции деления целых чисел. Источник задает делитель числа, делимое берется из регистра AX или регистровой пары DX:AX или регистровой пары EDX:EAX, частное помещается в регистр AX или EAX, остаток от деления – в регистры DX или EDX. Исходный код для печати частного в этом случае может быть записан так:

```
----------------------------------------
global PrintValue
PrintValue:
        mov rax,0
        mov rdx,0
        mov ax,10
        mov bx,2
        div bx
        ;mov ax,dx
        ret
----------------------------------------
```

Дополнительное задание: что произойдет с программой, если убрать строку mov rdx,0 ? перепишите данный пример с другими типами регистров, заменив AX на RAX или EAX, обоснуйте результат.

Пример. Написать программу для вычисления остатка от деления для выражения *(3+x*y)/(x-y)*, где *x, y* – заданные в программе целые числа.

```
----------------------------------------
global PrintValue
PrintValue:
        mov rax,0
        mov rdx,0
        mov bx,10
```

```
mov cx,2
mov ax,bx
mul cx
add ax,3
sub bx,cx
div bx
mov ax,dx
ret
```

Замечание. В данной программе в качестве *(x, y)* взяты *(10,2)*.

Дополнительное задание: удалите предпоследнюю команду mov AX,DX, как это изменение повлияло на результат? Поменяйте знак исходных данных.

3.1. Практические задания к разделу «Выполнение арифметических операций»

Используя приведенные ниже выражения, написать программу на ассемблере для вычисления результата. Рекомендуется взять небольшое число *n*, иначе программа получится громоздкой. Сокращать алгебраические выражения запрещено, нужно брать целую часть от результата деления.

1. Получить значение $Y=(1*X+1)*(2*X-4)*(3*X+7)*(4*X-10)* ... *(n*X-(3*n-2))$;
2. Получить значение $Y=6*2-4*4+2*6$;
3. Получить значение $Y=(1+3)*(3+5)*(5+7)* ... *((2*n-1)+(2*n+1))$.
4. Получить значение $Y=(1*2)+(3*4)+(5*6)+ ... +((2*n-a)*(2*n))$.
5. Получить значение $Y=2+4+6+ ... +2*n$.
6. Получить значение $Y=1+3+5+ ... +2*n+1$.
7. Получить значение $Y=1*1+2*2+3*3+...+n*n$.
8. Получить значение $Y=(1+3)*X+(2+4)*X+...+(n + (n + 2))*X$.
9. Получить значение $Y=1^3+2^3+3^3+...+n^3$.
10. Получить значение $Y=(1+3)*3-(5+7)*7- ... -(n + (n+2))*(n+2)$.
11. Получить значение $Y=1/3+3/5+5/7+ ... +(2*n-1)/(2*n+1)$.

12. Получить значение $Y=2/4+6/8+10/12+ \ldots +(2*n)/(2*n+2)$.
13. Получить значение $Y=5*2+7*5+\ldots+(2*n+3)*(3*n-1)$.
14. Получить значение $Y=5/1+7/2+\ldots+(2*n+3)/n$.
15. Получить значение $Y=1^2/2+2^2/3+\ldots+n^2/(n+1)$.
16. Получить значение $Y=1*2*3+4*5*6-7*8*9$.
17. Получить значение $Y=3/1+3/2+\ldots+3/n$.
18. Получить значение $Y=3*1+3*2+\ldots+3*n$.
19. Получить значение $Y=1^2/2^2+3^2/4^2+\ldots+(2*n-1)^2/(2*n)^2$.
20. Получить значение $Y=(2+1/2)+(2+2/2)+\ldots+(2+n/2)$.

4. Организация условных переходов

Для выполнения операции сравнения двух чисел в ассемблере имеется команда

```
CMP X,Y
```

Далее, в зависимости от ее результата выполняется команда условного перехода. Таким образом, данная команда изменяет значение регистра флагов. Можно использовать следующие команды выполнения условных переходов, например в зависимости от знака числа:

переходы при сравнении чисел *без знака*

```
JE метка  (X == Y)
JNE метка (X <> Y)
JB метка  (X < Y)
JAE метка (X >= Y)
JBE метка (X <= Y)
JA метка  (X > Y)
```

переходы при сравнении чисел *со знаком*

```
JE метка  (X == Y)
JNE метка (X <> Y)
JL метка  (X < Y)
JGE метка (X >= Y)
JLE метка (X <= Y)
JG метка  (X > Y)
```

Безусловный переход выполняется по команде:

```
JMP метка
```

Также для выполнения сравнения используют команду TEST, которая сама не производит дополнительных действий как CMP, а только изменяет флаги.

Пример. Написать программу для нахождения *min(x,y)*, где *x, y* – заданные в программе целые числа.

```
global PrintValue
section .data
x dd 40
y dd 35
PrintValue:
        mov rax,[x]
        mov rbx,[y]
        cmp eax,ebx
        jl g1
        mov rax,rbx
g1:     ret
```

Дополнительное задание: в команде cmp измените тип используемых регистров, например вместо EAX поставить AX или RAX. Как изменится результат после внесения изменений? Напишите аналогичную программу для нахождения *max(x,y)*.

Пример. Даны два числа *x, y* (с произвольным знаком) необходимо написать программу, заменяющую минимальное число на (*x * y),* а максимальное на *(3 * x – y)*.

```
global PrintValue
section .data
x dd -40
y dd 45
PrintValue:
        mov rax,[x]
        mov rbx,[y]
        cmp eax,ebx
        jl g1
        mov cx,3
        mul cx
        sub ax,bx
        mov rdi,rax
        mov rax,[x]
        mul rbx
        mov rbx,rax
        mov rax,rdi
        jmp end
```

```
g1:     mov cx,3
        mul cx
        sub ax,bx
        mov rbx,rax
        mov rax,[x]
        mov rcx,[y]
        mul rcx
end:    ret
```

Дополнительное задание: можно ли выполнить оптимизацию данного исходного кода по количеству используемых операторов? Как изменится результат выполнения программы, если заменить MUL RBX, MUL RCX на MUL BX, MUL CX?

4.1. Практические задания к разделу «Организация условных переходов»

Используя приведенные ниже выражения, написать программу на ассемблере для вычисления результата. Определить значение *z* в зависимости от значений *x,y*.

1. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} x^2 - xy + 3 \\ -x \end{pmatrix}$
2. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} -x + 4xy \\ 3x \end{pmatrix}$
3. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} x^3 - xy \\ -x + 3y \end{pmatrix}$
4. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} -xy + 5 \\ -x^2 y \end{pmatrix}$
5. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} -xy^2 + 1 \\ -x^2 y \end{pmatrix}$
6. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} (x-y)^2 \\ -x + 1 \end{pmatrix}$

7. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} (x+5)^2 - y \\ -xy+1 \end{pmatrix}$

8. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} xy^2 + x^2 y \\ -x+y \end{pmatrix}$

9. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} -x^3 + y^2 \\ -\frac{x}{y} \end{pmatrix}$

10. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} -xy-1 \\ x^2 - y^2 \end{pmatrix}$

11. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} -x^2 y + 7 \\ y(x-3) \end{pmatrix}$

12. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} \frac{(x-y)^2}{y} \\ x+y \end{pmatrix}$

13. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} x-5y \\ -\frac{y}{x}+1 \end{pmatrix}$

14. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} 5xy - y^2 \\ \frac{x+1}{y} \end{pmatrix}$

15. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} (x-2)(x+4) \\ -xy+1 \end{pmatrix}$

16. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} (x+y)(x+4) \\ y^2 + x^2 \end{pmatrix}$

17. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} (x-1)y^2 \\ (x-y)^2 \end{pmatrix}$

18. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} (-x+7)(-y-4) \\ -x(y+1) \end{pmatrix}$

19. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} (x-2y)^2 \\ -\dfrac{xy}{(x+y)} \end{pmatrix}$

20. $\begin{pmatrix} \min(x,y) \\ \max(x,y) \end{pmatrix} := \begin{pmatrix} 3y(x-2) \\ x+y^2 \end{pmatrix}$

5. Циклы со счетчиком

Для того, чтобы организовать цикл, выполняющийся заданное количество раз, в ассемблере используется команда

```
LOOP метка
```

Метка должна быть указана до использования самой команды. Таким образом, получается, что данный цикл всегда выполняется минимум один раз. Общее число итераций должно быть предварительно записано в регистр CX. После выполнения очередной итерации это содержимое регистра уменьшается на единицу. Выполнение цикла останавливается при условии CX=0.

В качестве примера приведем классическую задачу вычисления функции факториал.

Пример. Задано число *N*, написать программу для вычисления *N!=1*2*3…N*. Для реализации рекомендуется брать *N* не более 20.

```
----------------------------------------
global PrintValue
section .data
N dq 12
PrintValue:
        mov rcx,[N]
        mov rax,1
        mov rsi,1
begin:  mul rsi
        inc rsi
        loop begin
        ret
----------------------------------------
```

Замечание. В данной программе используется числовой тип DQ (QWORD) размером 64 бит.

Дополнительное задание: в программе попробуйте использовать другие типы данных. Как изменится результат после внесения изменений? Почему при выборе больших значений N результат становится неверным?

5.1. Практические задания к разделу «Циклы со счетчиком»

Используя приведенные ниже выражения, написать программу на ассемблере для вычисления суммы конечного числового ряда.

1. $\sum_{n=1}^{5} n!$
2. $\sum_{n=1}^{5} (2n)!$
3. $\sum_{n=1}^{5} (2n-1)!$
4. $\sum_{n=1}^{5} (2n)!+2$
5. $\sum_{n=1}^{5} n!+4$
6. $\sum_{n=1}^{5} 2^n$
7. $\sum_{n=1}^{5} n^n$
8. $\sum_{n=1}^{5} (n+2)!$
9. $\sum_{n=1}^{5} 4^n$
10. $\sum_{n=1}^{5} n!+n$

11. $\sum_{n=1}^{5} 2n!$

12. $\sum_{n=1}^{5} (2n+3)!$

13. $\sum_{n=1}^{5} 2^n + n$

14. $\sum_{n=1}^{5} (3n)!$

15. $\sum_{n=1}^{5} 2^{n+1} + 1$

16. $\sum_{n=1}^{5} n^2 + n!$

17. $\sum_{n=1}^{5} (6-n)^n$

18. $\sum_{n=1}^{5} (7-n)!$

19. $\sum_{n=1}^{5} 3^n - 5$

20. $\sum_{n=1}^{5} 20 - (2n)!$

6. Обработка числовых массивов

Описание линейного массива в программе возможно с инициализацией путем явного перечисления значений элементов, либо без нее, указав лишь тип и количество элементов массива. Например:

```
Array dd 1,2,3,4,5,6,7
```

либо через использование секции неинициализированных данных или секции *BSS*, обозначающейся в исходном коде «.bss»

```
section .bss
Array resd 7
```

При написании программ обработки таких массивов необходимо вычислять номер искомого элемента через базовый адрес массива плюс смещение от него на необходимый индекс умноженный на размер элемента в байтах.

Довольно часто приходится обращаться к процедуре определения адреса нужного элемента массива. В этом помогает команда LEA, которая не обращаясь к памяти, «загружает» искомый адрес в регистр, например

```
lea rax,Array
```

Пример. Вычислить сумму элементов массива, состоящего из заданных целых чисел.

```
global PrintValue
section .data
Array dd 1,2,3,4,5,6,7
Sum dd 0
PrintValue:
        mov rax,0
        mov rcx,7
        mov rsi,0
begin:  add rax,[Array + rsi]
        add rsi,4
        loop begin
        mov [Sum],rax
        ret
```

Замечание. В данной программе используется переменная Sum, в которую записывается значение регистра RAX, являющееся суммой элементов массива.

Следующий пример в своей основе использует код из книги Столярова А.В. [1]

Пример. Заполнить неинициализированный линейный массив, состоящий из 255 числовых элементов типа DWORD (32 бит), нулями.

```
global PrintValue
section .bss
Array resd 255
section .text
```

```
PrintValue:
        mov rcx,255
        xor rax,rax
        mov rdi,Array
        cld
begin:  stosd
        loop begin
        mov rax,[Array + 4]
        ret
_______________________________________
```

Замечание. В данном примере используются следующие команды: STOSD – записывает в память по адресу [RDI] двойное слово из регистра RAX и увеличивает регистр RDI на четыре, CLD – сброс флага DF для установки увеличения регистра RDI, а не его уменьшения (что тоже возможно), директива RESD служит для резервирования памяти под наш массив из 255 элементов.

Пример. Дана числовая матрица 4x4. Найти сумму элементов над главной диагональю.

```
_______________________________________
global PrintValue
section .data
Array dd 1,2,3,4, 2,3,4,5, 3,4,5,6, 4,5,6,7
Sum dd 0
PrintValue:
        mov rax,0
        mov rcx,3
        mov rbx,4
begin:  mov rdx,rcx
        mov rsi,rbx
begin1: add rax,[Array + rsi]
        add rsi,4
        loop begin1
        add rbx,20
        mov rcx,rdx
        loop begin
        mov [Sum],rax
        ret
_______________________________________
```

Замечание. Двухмерная матрица хранится в памяти как обычная числовая последовательность, то есть элементы идут

подряд друг за другом, поэтому необходимо уметь вычислять начало следующей строки в байтах смещением от ее начала.

Дополнительное задание: попробуйте увеличить размерность матрицы. Что необходимо изменить в данной программе?

6.1. Практические задания к разделу «Обработка числовых массивов»

1. а) Задан линейный массив *N* из элементов. Определить количество элементов, являющихся четными числами.
1. б) Дана матрица *N*x*N*. Найти сумму элементов главной и побочной диагонали.
2. а) Задан линейный массив *N* из элементов. Определить сумму элементов, являющихся четными числами.
2. б) Дана матрица *N*x*N*. Найти сумму максимального элемента и его индексов.
3. а) Задан линейный массив *N* из элементов. Определить среднее арифметическое значение элементов, являющихся четными числами.
3. б) Дана матрица *N*x*N*. Найти произведение индексов максимального элемента.
4. а) Задан линейный массив *N* из элементов. Определить произведение элементов, являющихся четными числами.
4. б) Дана матрица *N*x*N*. Найти сумму элементов четных строк.
5. а) Задан линейный массив *N* из элементов. Определить количество элементов, имеющих четные порядковые номера и являющихся четными числами.
5. б) Дана матрица *N*x*N*. Найти сумму элементов нечетных строк.
6. а) Задан линейный массив *N* из элементов. Определить количество элементов, имеющих нечетные порядковые номера и являющихся четными числами.
6. б) Дана матрица *N*x*N*. Найти сумму элементов четных столбцов.
7. а) Задан линейный массив *N* из элементов. Определить количество элементов, имеющих четные порядковые номера и являющихся нечетными числами.

7. б) Дана матрица *N*x*N*. Найти сумму элементов нечетных столбцов.
8. а) Задан линейный массив *N* из элементов. Определить количество элементов, имеющих нечетные порядковые номера и являющихся нечетными числами.
8. б) Дана матрица *N*x*N*. Найти сумму максимальных элементов столбцов.
9. а) Задан линейный массив *N* из элементов. Определить количество элементов, кратных 3.
9. б) Дана матрица *N*x*N*. Найти сумму максимальных элементов строк.
10. а) Задан линейный массив *N* из элементов. Определить количество элементов, не кратных 5.
10. б) Дана матрица *N*x*N*. Найти сумму с четной суммой индексов.
11. а) Задан линейный массив *N* из элементов. Определить количество элементов, являющихся отрицательными и нечетными.
11. б) Дана матрица *N*x*N*. Найти количество элементов с нечетной суммой индексов.
12. а) Задан линейный массив *N* из элементов. Определить количество элементов, являющихся положительными и нечетными.
12. б) Дана матрица *N*x*N*. Найти количество строк, в которых четных элементов больше нечетных.
13. а) Задан линейный массив *N* из элементов. Определить количество элементов, являющихся положительными и нечетными.
13. б) Дана матрица *N*x*N*. Найти количество столбцов, в которых нечетных элементов больше четных.
14. а) Задан линейный массив *N* из элементов. Определить количество элементов, отличающихся от последнего.
14. б) Дана матрица *N*x*N*. Для заданного числа *X* найти сумму элементов, сумма индексов которых кратна *X*.
15. а) Задан линейный массив *N* из элементов. Определить количество элементов, совпадающих с первым элементом.
15. б) Дана матрица *N*x*N*. Для заданного числа *X* найти количество элементов, сумма индексов которых меньше *X*.

16. а) Задан линейный массив *N* из элементов. Определить количество элементов, меньше заданного числа *X*.
16. б) Дана матрица *N*x*N*. Найти сумму элементов главной и побочной диагонали.
17. а) Задан линейный массив *N* из элементов. Определить количество элементов, больше заданного числа *X*.
17. б) Дана матрица *N*x*N*. Найти сумму элементов главной и побочной диагонали.
18. а) Задан линейный массив *N* из элементов. Определить количество элементов, равных заданному числу *X*.
18. б) Дана матрица *N*x*N*. Найти сумму элементов главной и побочной диагонали.
19. а) Задан линейный массив *N* из элементов. Определить количество элементов, являющихся степенями двойками.
19. б) Дана матрица *N*x*N*. Найти сумму элементов главной и побочной диагонали.
20. а) Задан линейный массив *N* из элементов. Определить сумму элементов, равных заданному числу *X*.
20. б) Дана матрица *N*x*N*. Найти сумму элементов главной и побочной диагонали.

7. Организация и вызов подпрограмм

Для улучшения структуры исходного кода часто используют подпрограммы, в которые оформляют наиболее часто повторяющиеся его фрагменты. Подпрограмма имеет собственное имя, по которому к ней можно обращаться. В подпрограммы можно передавать необходимые данные через параметры. При их передаче лучше использовать стек данных – это специальная область памяти, создаваемая операционной системой при запуске приложения, работающая по принципу LIFO – «*last in first out*».

Адрес-указатель на вершину стека находится в регистре RSP, и его значение меняется при добавлении или удалении данных из стека. Для того, чтобы не было конфликтов при вызовах различных подпрограмм, работающих со стеком, при очередном вызове необходимо сохранить значение RSP например в RBP

предварительно сохранив его в стеке. Перед выходом из подпрограммы необходимо вернуть сохраненные значения регистров. Например

```
push rbp
mov rbp,rsp
mov rsp,rbp
pop rbp
ret
```

Итак, вызов подпрограммы идет с помощью команды CALL, возврат из нее – RET.

Пример. Даны два числа – *X* и *Y*, вычислить *min(X,Y)*.

```
global PrintValue
section .data
x dd 40
y dd 45
PrintValue:
        mov rax,[x]
        push rax
        mov rax,[y]
        push rax
        call min
        pop rbx
        pop rbx
        ret
min:
        push rbp
        mov rbp,rsp
        mov rax,[rbp+16]
        mov rbx,[rbp+24]
        cmp eax,ebx
        jl g1
        mov rax,rbx
g1:     mov rsp,rbp
        pop rbp
        ret
```

Дополнительное задание: напишите программу для вычисления максимума двух чисел, вычисления дискриминанта для решения квадратного уравнения.

7.1. Практические задания к разделу «Организация и вызов подпрограмм»

1. Даны три целых числа *x, y, z*, требуется найти *max(x,y+z) - min(z,x-y)*.
2. Даны три целых числа *x, y, z*, требуется найти *max(2*x,z) - min(z,2*y)*.
3. Даны три целых числа *x, y, z*, требуется найти *max(x,y) + max(y,z) + max(x,z)*.
4. Даны два целых числа *x, y,* требуется найти *f(2x,x+y)*, где *f(s,t)=(s+t)(s-2t)*.
5. Даны два целых числа *x, y,* требуется найти *f(3x,5y)*, где *f(s,t)=(s-t)(3s-2t)*.
6. Даны два целых числа *x, y,* требуется найти *f(y-x,x-y)*, где *f(s,t)=2st(3s-2t)*.
7. Задан массив из пяти целых чисел, необходимо найти сумму квадратов этих чисел, вычисление квадрата числа оформить в виде процедуры.
8. Задан массив из пяти целых чисел *A1…A5*, необходимо найти сумму чисел *B1…B5*, где *Bi=A1+A2+…Ai*, вычисление которого оформить подпрограммой.
9. Даны два целых числа *x, y,* требуется найти $max(k+m,m^2)$, где *k=min(x,y), m=max(x,y)*.
10. Даны два целых числа *x, y,* требуется найти $max(km,m^2)$, где $k=min(x+2y,y^2)$, $m=min(x^2,y)$.
11. Даны два целых числа *x, y,* требуется найти *min(k-m,m+2k)*, где $k=min(x^2+2y,xy^2)$, $m=min(x^2,y^2)$.
12. Даны пять целых чисел *A1…A5*, требуется найти сумму $f(A1)+…+f(A5)$, *где* $f(X)=X^2+2X-1$.
13. Даны пять целых чисел *A1…A5*, требуется найти сумму $f(A1)+…+f(A5)$, *где* $f(X)=X^3-X^2-3$.
14. Даны пять целых чисел *A1…A5*, требуется найти сумму $f(A1)+…+f(A5)$, *где* $f(X)=2X^2+3X-1$.

15. Даны два числа *n,k (k<n)*, вычислить значение $C_n^k = \frac{n!}{k!(n-k)!}$, где вычисление факториала оформить отдельной процедурой.
16. Даны пять целых чисел *A1…A5*, получить произведение всех отрицательных. Для проверки оформить отдельную процедуру.
17. Даны пять целых чисел *A1…A5*, получить произведение всех четных. Для проверки оформить отдельную процедуру.
18. Даны шесть чисел *A1…A6*, получить сумму всех пар чисел, отличающихся по модулю на единицу. Для проверки оформить отдельную процедуру.
19. Даны шесть чисел *A1…A6*, получить сумму чисел, являющихся квадратами чисел 1,3,5,7. Для проверки оформить отдельную процедуру.
20. Даны шесть чисел *A1…A6*, получить сумму всех пар чисел, отличающихся по модулю на *min(A1,…,A6)*. Для проверки оформить отдельную процедуру.

8. Основы работы в графической среде sasm

Согласно информации из Википедии *SASM* (*SimpleASM*) — бесплатная простая кроссплатформенная *Open Source* среда разработки программного обеспечения на языках ассемблера *NASM, MASM, GAS, FASM* с подсветкой синтаксиса и отладчиком.

SASM позволяет легко разрабатывать и выполнять программы, написанные на языке ассемблера. Программа работает «из коробки» и хорошо подходит для начинающих изучать язык ассемблера.

Основана на Qt. Распространяется по свободной лицензии GNU GPL v3.0. Создана программистом Дмитрием Манушиным (https://dman95.github.io/SASM/).

В программу включена библиотека макросов для *NASM* "**io.inc**". В ней есть кроссплатформенные команды ввода-вывода и макросы: CMAIN - точка входа и CEXTERN для доступа к внешним функциям на языке C.

Для работы в графической среде Linux под ОС Windows воспользуемся программным продуктом *Xmanager* (https://www.netsarang.com/).

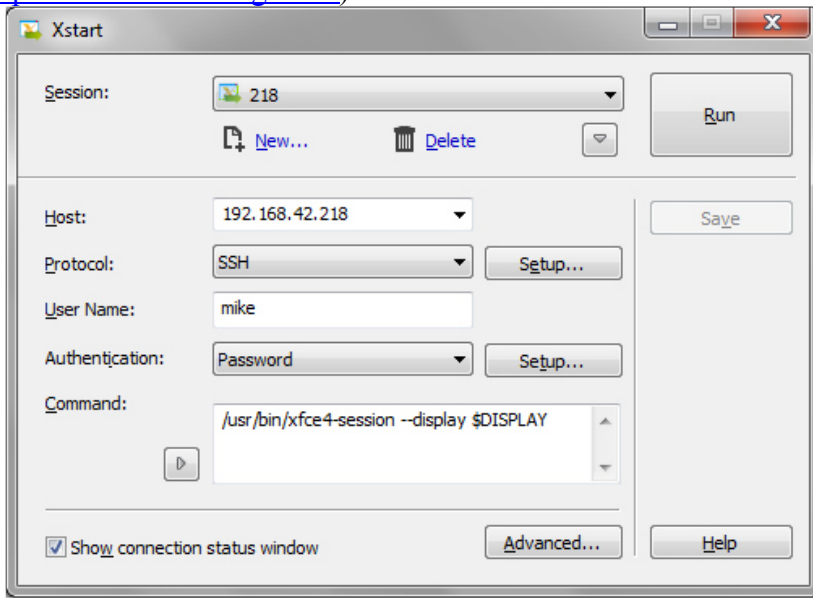

Рис. 4. Настройки соединения **Xmanager.**

При запуске важно указать: Host (192.168.42.218), User name (Логин для входа в систему), Command (/usr/bin/xfce4-session –display $DISPLAY) для старта графической оболочки Xface.

В «Меню приложений» - «Разработка» выбрать «Sasm», должно открыться следующее окно

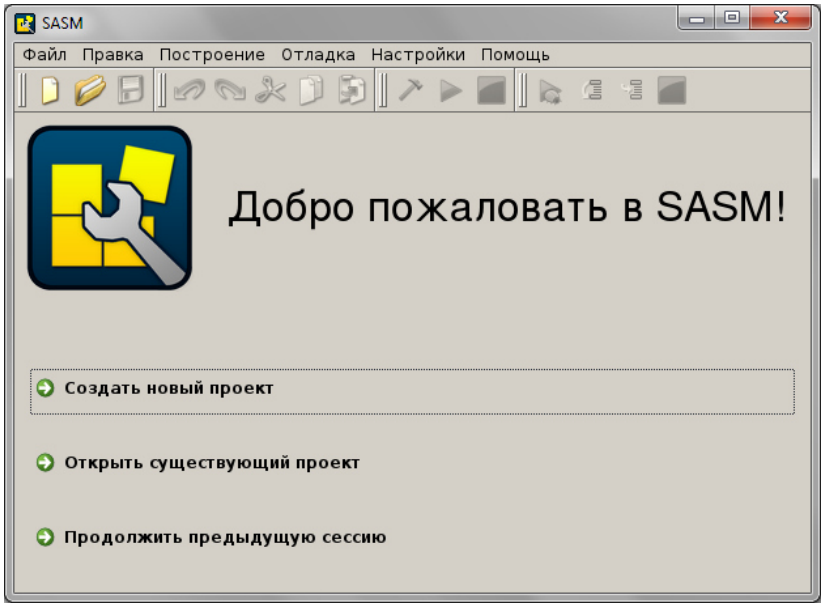

Рис. 5. Главное окно **Sasm.**

В исходном коде можно использовать следующие макросы:

PRINT_CHAR ch — печатается символ, заданный параметром ch. В качестве параметра может выступать численная константа, символьная константа, имя переменной, имя регистра или адресное выражение (без спецификатора размера данных в памяти). Печатается всегда содержимое 8 младших разрядов;

PRINT_STRING data — печать строки текста, оканчивающейся символом с кодом 0. В качестве параметра можно передавать строковую константу, имя переменной или адресное выражение (без спецификатора размера данных в памяти). В случае печати строковой константы, наличие символа с кодом 0 в конце строки необязательно;

NEWLINE — макрос переводит печать на новую строку;

GET_CHAR data — происходит считывание одного символа, нажатие Enter не требуется. Более того, нажатие Enter будет расцениваться как ввод управляющих символов перевода строки: 0xD 0xA в ОС Windows, 0xA в ОС *nix. Если параметр – регистр, размер которого больше 1 байта, значение считанного символа будет дополнено нулями.

GET_STRING data, maxsz — ввод последовательности символов длиной не более чем (maxsz-1). Чтение последовательности останавливается на EOF или переводе строки, причем перевод строки сохраняется в буфере. В конец считанной строки добавляется символ с кодом 0. Параметр data – либо имя переменной, либо адресное выражение (без спецификатора размера данных в памяти). Параметр maxsz – регистр или числовая константа.

9. Работа с вещественными числами

Арифметический сопроцессор имеет 80-битовых регистров для хранения вещественных чисел, которые обозначим R0, R1,…,R7; регистры образуют стек, то есть один из регистров Rn считается вершиной стека и обозначается в программе ST0, следующий за ним ST1 и т.д., причем считается, что следом за R7 идет R0 (например, если R7 в настоящий момент обозначен как ST4, то по факту роль ST5 будет играть регистр R0). Роль вершины стека может играть любой из регистров Rn, причем при занесении нового значения в этот стек все значения, которые там уже хранились, остаются на своих местах, а меняется только номер регистра, играющего роль вершины.

Также имеется свой регистр состояния SR (state register), содержащий ряд флагов, описывающих состояние арифметического сопроцессора, находящегося внутри центрального процессора.

В частности, биты 13,12,11 содержат число от 0 до 7, называемое TOP и показывающее, какой из регистров Rn в настоящий момент считается вершиной стека. Флаги C0 (8 бит), C2 (10 бит) и C3 (14 бит) соответствуют по сути флагам центрального процессора CF, PF и ZF.

Мнемонические обозначения всех машинных команд, имеющих отношение к арифметическому сопроцессору, начинаются с буквы f от английского floating.

В качестве операнда могут выступать регистры сопроцессора, обозначаемые STn, либо операнды типа «память».

Операнд типа «память» может быть только четырехбайтным (dword, dw), восьмибайтным (qword, dq) и десятибайтным (tword, dt).

Определены следующие команды передачи данных в вещественном формате, аналогичны, по сути команде MOV:

`fld источник` – загрузка вещественного числа из области памяти (источник) в вершину стека сопроцессора (ST0);

`fst приемник` – сохранение вещественного числа из вершины стека в память;

`fstp приемник` – то же, но с выталкиванием числа из стека;

`fxch st`**`i`** – обмен значений между вершиной стека и регистром стека сопроцессора.

Замечание: таким образом, действие команды FLD можно сравнить с командой PUSH. Аналогичные команды для целых чисел просто содержат букву i после буквы f.

Определены следующие арифметические команды:

Операция сложения

`fadd` – сложение st**0** и st**1** и результат в st**0**;

`fadd источник` – сложение st**0** и источника, результат в st**0**;

`fadd st`**`i`**`,st0` – сложение st**i** и st**0**, результат в st**0**;

`faddp st`**`i`**`,st0` – сложение st**i** и st**0**, результат в st**i-1**, а из st**0** число выталкивается;

Операция вычитания

`fsub` – вычитает значение st**1** из значения st**0**, результат в st**0**;

`fsub источник` – вычитает значение источника из значения st**0**, результат в st**0**;

`fsub st`**`i`**`,st` – вычитает значение st**0** из значения st**i**, результат в st**i**;

`fsubp st`**`i`**`,st0` – вычитает значение st**0** из значения st**i**, результат в st**i-1**, причем из st**0** значение выталкивается;

Операция умножения

`fmul` – умножает st**0** на st**1**, результат в st**0**;

`fmul st`**`i`** – умножает st**0** на st**i**, результат в st**0**;

`fmul st`**`i`**`,st0` – умножает st**0** на st**i**, результат в st**i**;

`fmulp sti,st0` – умножает st**0** на st**i**, результат в st**i-1**, из st**0** значение выталкивается;

Операция деления

`fdiv` – делит st**0** на st**1**, результат в st**0**;

`fdiv sti` – делит st**0** на st**i**, результат в st**0**;

`fdiv sti,st0` – делит st**0** на st**i**, результат в st**i**;

`fdivp sti,st0` – делит st**0** на st**i**, результат в st**i-1**, из st**0** значение выталкивается.

Пример. Поместить в регистры стека два вещественных числа и сложить их, результат вывести на экран. Для отладки программы использовать пакет sasm.

```
----------------------------------------
%include "io.inc"
section .data
f1 dd 17.123
f2 dd 109.931
res dd 0
section .text
global CMAIN
CMAIN:
    finit
    fld dword[f1]
    fld dword[f2]
    fadd
    mov eax,4
    push eax
    call OutFloat
    pop eax
    ret
OutFloat:
        enter   4, 0 ; пролог - выделим в кадре стека
4 байта под локальные переменные
        mov     dword[ebp-4], 10
        ftst ; определяем знак числа
        fstsw   ax
        sahf
        jnc     @positiv
        mov     al, '-' ; если число отрицательное -
выводим минус
        PRINT_CHAR al
        fchs ; и получаем модуль числа
```

```
  @positiv:
          fld1 ; загружаем единицу
          fld     st1 ; копируем число на вершину стека
          fprem ; выделим дробную часть
          fsub    st2, st0 ; отнимем ее от числа - по-
лучим целую часть
          fxch    st2 ; меняем местами целую и дробную
части
          xor     ecx, ecx ; обнуляем счетчик
  ; далее идет стандартный алгоритм вывода целого числа
на экран
          push    dword[ebp-4]
  @1:
          fidiv   dword[ebp-4] ; делим целую часть на
десять
          fxch    st1 ; обменяем местами st0 и st(1)
для команды fprem
          fld     st1 ; копируем результат на вершину
стека
          fprem ; выделим дробную часть (цифру справа
от целой части)
          fsub    st2, st0 ; получим целую часть
          fimul   dword[ebp-4] ; *10
          fistp   dword[ebp-8] ; получаем очередную
цифру
          push    dword[ebp-8] ; заталкиваем ее глубже
в стек
          ;mov edx,dword[ebp-8]
          inc     ecx ; и увеличим счетчик
          fxch    st1 ; подготовим стек к следующему
шагу цикла (полученное частное на вершину, в st(1) - 1)
          ftst ; проверим не получили ли в частном 0?
          fstsw   ax
          sahf
          jnz     @1 ; нет - продолжим цикл
  @2: ; извлекаем очередную цифру, переводим её в сим-
вол и выводим.
          pop     eax
          add     al, '0'
          PRINT_CHAR al
          loop    @2
          pop     eax
```

```
   ; далее то же самое, только для дробной части. Алго-
ритм похож на вывод целого числа, только вместо деления
умножение и проход по числу слева
          fstp    st0 ; сначала проверим, есть ли дроб-
ная часть
          fxch    st1
          ftst
          fstsw   ax
          sahf
          jz      @quit ; дробная часть отсутствует
          mov     al, '.'
          PRINT_CHAR al ; если присутствует - выведем
точку
          mov     ecx, [ebp+8] ; помещаем в счетчик
длину дробной части
   @3:    fimul   dword[ebp-4] ; умножим на 10
          fxch    st1           ; подготовка для fprem
- меняем st0 и st(1) местами и
          fld     st1 ; копируем число на вершину
          fprem ; отделим дробную часть от целой
          fsub    st2, st0 ; и оставляем дробную
          fxch    st2
          fistp   dword[ebp-8] ; выталкиваем полученное
число из стека в temp
          mov     ax, [ebp-8] ; по дробной части идем
слева, значит число выводим сразу, без предварительного
сохранения в стек
          or      al, 30h ; перевод в ascii
          call Print ; на экран
          fxch    st1 ; подготовим стек к следующему
шагу цикла (полученное частное на вершину, в st(1) - 1)
          ftst
          fstsw   ax
          sahf ; проверим на 0 остаток дробной части
          loopne  @3
   @quit:
          fstp ; готово. Чистим стек сопроцессора
          fstp    st0
          leave ; эпилог
          ret
   Print:
       PRINT_CHAR al
       ret
```

_ _

Замечание: большая часть данной программы занимает процедура печати вещественного числа на экран (источник http://samplecode.ru/?a=p&i=1347), также используются макросы вывода символа на экран из стандартной библиотеки **"io.inc"**.

Дополнительное задание: оформите процедуру OutFloat в отдельный файл, чтобы стало возможным ее использование как макросом.

9.1. Практические задания к разделу «Работа с вещественными числами»

1. Вычислить $\dfrac{7{,}45+\sqrt{3{,}45^2+7{,}61}}{12{,}11+3{,}8*9{,}1}$, $\dfrac{1{,}23^2-6{,}74}{4*(\sin(3)+\cos(1{,}11))}$;
2. Вычислить $\dfrac{-3{,}12*\sqrt{2{,}05^2-1{,}61}}{11{,}11+3{,}3*3{,}1}$, $\dfrac{\cos(1{,}23\pi)-3{,}14}{4{,}5*(\sin(1{,}34)-\cos(0{,}31))}$;
3. Вычислить $\sqrt{4{,}35^2-3{,}71}-\dfrac{7{,}45+8{,}78}{7{,}32*3{,}17}$, $\dfrac{\sin(1{,}45^2)}{4{,}3*5{,}5}$;
4. Вычислить $\sqrt{6{,}71+3{,}21^2}+\dfrac{\sqrt[3]{67{,}6}}{4{,}35-7{,}91}$, $\dfrac{\sin(0{,}78)+\cos(0{,}18)}{1{,}34^2}$;
5. Вычислить $6{,}76^3-\dfrac{7{,}45+\sqrt{76{,}77}}{5{,}45}$, $\dfrac{43*\sin\left(\dfrac{\pi}{3}\right)}{4*\sqrt{3{,}12}}$;
6. Вычислить $\dfrac{\sqrt{29{,}3}+3{,}45^2}{13{,}31+1{,}77*19{,}1}$, $\dfrac{\cos^2(3{,}31)}{4*\sqrt[3]{54*\sin^2(5{,}67)}}$;
7. Вычислить $4{,}77-\dfrac{17{,}65^3}{12{,}11+\sqrt{10{,}45}}$, $\dfrac{39{,}11-ctg(67{,}9)}{2{,}33*tg^2(55{,}17)}$;

8. Вычислить $\dfrac{7,45*8,97}{10,01-12,7*9,19}-\sqrt{\dfrac{4,5^2+3,61}{7,71}}$,

$\sin(0,25)*\cos(0,25)-\dfrac{tg^2(9,9)}{0,77*\sin(0,75)*\cos(0,75)}$;

9. Вычислить $\sqrt[3]{\dfrac{77,96}{0,71}}+\dfrac{6,1+3,34-2,14}{2,18*1,91}$,

$\sqrt{\sin^2\left(\dfrac{\pi}{12}\right)+1}+\dfrac{\sin\left(\dfrac{\pi}{12}\right)}{0,17*\left(\dfrac{\pi}{12}+1\right)}$;

10. Вычислить $\dfrac{13,45^2+\sqrt{123,31}}{13,81*4,41-2,27}$, $tg^2(3,37)+\dfrac{7,03^2+61,64}{\sqrt[3]{9,97}}$;

11. Вычислить $\dfrac{\sqrt{37,73}-2,67^2}{0,15+0,38*0,91}$, $\dfrac{\sqrt{34,17^2-\sqrt[3]{113,71}}}{\cos(0,24)}$;

12. Вычислить $\dfrac{\sqrt{347,33-3,1^3}}{5,18*7,19}$, $\dfrac{3,45^3}{\sin(1)}-\dfrac{tg^2(1)}{\cos(3,73)}$;

13. Вычислить $\dfrac{5^{2,5}+3^{3,5}}{\sqrt{45,7^2-4,1*6,8}}$,

$\sin(3,4)*\cos(5,6)-\sin(2,1*5,7)*\cos(1,12*9,9)$;

14. Вычислить $\dfrac{\sqrt[3]{55,77*99,91}*\sqrt{78,7-4,33}}{3,12*6,15}$,

$14,3*\left(\dfrac{\cos^2(1,17)-\sin(2,13)}{tg\left(\dfrac{4,67}{3,31}\right)}-\dfrac{\pi}{11}\right)$;

15. Вычислить $67{,}33^3 - 5{,}93^2 + 67{,}33 * 5{,}93$,

$tg^2\left(\frac{7{,}75}{3{,}31}\right) * \cos^2(7{,}75 * 3{,}31)$;

16. Вычислить $(5{,}71 + 4{,}43)^3 - \frac{\sqrt{5{,}35^3 - 2{,}1^2}}{5{,}71 * 4{,}43}$,

$tg^2(\cos(5{,}75) + 10 * \sin(3{,}75))$;

17. Вычислить $\sqrt{7{,}33 * 23{,}13^3}\,\frac{\sqrt{6{,}78 + 5{,}79}}{\sqrt{6{,}78 - 5{,}79}}$, $\sqrt{tg^2\left(\frac{\cos(7{,}99^3)}{\sin(7{,}99^2)}\right)}$;

18. Вычислить $\frac{(345{,}1 - 243{,}4)^2}{3{,}18 * 2{,}29 + 7{,}01 * 9{,}97}$,

$(23{,}3 * \cos(3{,}04) + 7{,}97 * \sin(3{,}04))^2$;

19. Вычислить $\frac{\sqrt{45{,}47 * 9{,}91} * \sqrt{45{,}47 - 2 * 9{,}91}}{3{,}18 * 16{,}16 + 79{,}81}$,

$\frac{tg^3(3{,}56 * 7{,}87)}{2{,}13 * \cos^2\left(\frac{1{,}12}{0{,}13}\right)}$;

20. Вычислить $\frac{\sqrt{98{,}98^3 - 32{,}7 * 5{,}79}}{\sqrt[3]{3{,}33}}$,

$\sin^3\left(\frac{6{,}97^2}{4{,}9 - 3{,}2}\right) * tg\left(\sqrt{69{,}3 * 78{,}7}\right)$.

Список литературы

1. Столяров А.В. Программирование на языке ассемблера NASM для ОС Unix: Уч. пособие. М.: МАКС Пресс, 2011. 188 с.

2. Шелупанов А.А., Кирнос В.Н. Информатика. Базовый курс. Часть 1. Общие вопросы информатики и программирование на Ассемблере. Томск: ТУСУР, 2007. 190с.

3. Пильщиков В.Н. Программирование на языке ассемблера IBM PC : Учеб.пособие. - М.: ДИАЛОГ-МИФИ, 2000. - 286с.

4. Магда Ю.С. Ассемблер. Разработка и оптимизация Windows-приложений - СПб.: БХВ-Петерб., 2003. - 540с.

5. Пирогов В.Ю. Ассемблер для Windows. - 3-е изд. - СПб. : БХВ-Петерб., 2005. - 844 с.

СОДЕРЖАНИЕ

Учебное издание

Михаил Аркадьевич Клочков

Программирование на языке ассемблер в ОС Linux

Учебно-методическое пособие

Авторская редакция

Отпечатано с оригинал-макета заказчика
Подписано в печать 1.02.18. Формат 60x84$^{1/16}$.
Печать офсетная. Усл. печ. л. Уч.-изд. л.
Заказ № Тираж 30 экз.

Издательство «Удмуртский университет»
426034, Ижевск, Университетская, д. 1, корп. 4, каб. 207
Тел./факс: + 7 (3412) 500-295. E-mail: editorial@udsu.ru