Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

«Языки системного программирования: язык Ассемблера»

Проверил:
Сентерев Ю.А _____
«_____» _____ 201_г.

Выполнил:
Студент группы P3255
Федюкович С. А _____

Оценка _____

Санкт-Петербург

2018

Оглавление

Цель работы

Закрепление знаний и овладение навыками разработки программ для аппаратно-программных комплексов на языке Ассемблера.

Введение

Ассемблер (Assembly) — язык программирования, понятия которого отражают архитектуру электронно-вычислительной машины. Язык ассемблера — символьная форма записи машинного кода, использование которого упрощает написание машинных программ.

Задача 1

Рассмотрите следующий код и приведите краткое описание каждой строки программы:

```
section .text
global _start
 _start:
mov ecx, message
mov edx, length
mov ebx, 1
mov eax, 4
int 0x80
mov eax, 1
int 0x80
section .data
message db ' !', 10 Ассемблер рулит
length equ $ - message
```

Ответ

Приведем код программы с комментариями:

```
section .text ; директива ассемблера о том что приведеннный ниже код д.б.
 ↪  расположен в секции кода text финального исполняемого файла;
global _start ; директива, сообщающая с какой точки должно начаться исполнение
 ↪  файла
 _start: ; метка, к которой можно обращаться в коде
mov ecx, message ; копирование строки расположенной по адресу message в регистр
 ↪  процессора ecx
mov edx, length ; копирование числа, представляющего длину строки message в регистр
 ↪  процессора edx
mov ebx, 1 ; занесение в регистр процессора ebx числа 1, обозначающего дескриптор
 ↪  стандартного потока вывода
mov eax, 4 ; занесение в регистр eax числа 4, соответствующего коду системного
 ↪  вызова sys_write (Запись данных в дескриптор файла)
int 0x80 ; передача управления ядру ОС - interrupt (прерывание)
mov eax, 1 ; занесение в регистр eax числа 1, соответствующего системному вызову
 ↪  sys_exit
int 0x80 ; передача управления ядру ОС - interrupt (прерывание)
section .data ; директива ассемблера о том, что код ниже д. б. расположен в секции
 ↪  кода data финального исполняемого файла
message db ' !', 10 Ассемблер рулит ; метка message; db - параметр, указывающий на
 ↪  то, что вместо метки message в коде должно быть размещено несколько байт; 10 -
 ↪  перенос строки
```

```
length equ $ - message ; метка length с параметром equ чтобы сообщить, что эта
↪  метка является эквивалентом чего-либо; символ "$" - соотв. текущей позиции в
↪  коде т.о. метка length обозначает текущую позицию за вычетом расположения
↪  строки с меткой message
```

Задача 2

Объясните, что такое дизассемблирование кода программы и когда оно применяется. Можно ли провести дизассемблирование кода программ, разработанных с использованием других языков программирования, приведите пример.

Ответ

Дизассемблирование кода программы это процесс преобразования инструкций центрального процессора в их текстовые эквиваленты на одном из языков ассемблера. При этом можно получить список инсрукций из секции кода бинарного файла. Дизассемблирование применяется когда требуется посмотреть на детали реализации программы, написанной кем-то другим. Например, для ознакомления с реализацией критичных к времени исполнения функций и т. д. Можно провести дизассемблирование кода программ, разработанных на других языках программирования, таких как C. Однако, необходимо понимать, что получить текст исходной программы на C невозможно. Результатом дизассемблирования будет код, сгенерированный компилятором из исходного кода программы.

Задача 3

Оцените возможности языка Ассемблера с точки зрения производительности готовых к выполнению программ и эффективности разработки драйверов устройств.

Ответ

В некоторых случаях код написанный на языке ассемблера может работать быстрее кода, сгенерированного компилятором. Язык ассемблера целесообразно применять условиях ограниченных ресурсов (объёма носителя данных) т. к. исполняемые файлы, получаются очень компактными. Так, ассемблер широко используется для программирования встраеваемых устройств. Однако написание сложных программ для современных настольных ПК на языках ассемблера является нецелесообразным из-за сложности портирования на другие архитектуры ЦП, Кроме того процесс отладки сложнее, чем у программ, написанных на ЯВУ, а код более запутанный. Многие драйверы до сих пор пишутся на языке ассемблера, т. к. он являются лучшим языком программирования для непосредственного взаимодействия с аппаратным обеспечением, а также из соображений быстродействия.

Задача 4

Приведите последовательность действий и инструменты, которые при этом используются для преобразования кода программы на языке ассемблера в исполняемый бинарный файл.

Ответ

Весь процесс называется компиляцией и он состоит из двух этапов. Сначала происходит ассемблирование — генерация объектного кода, при этом создаётся файл с расширением *.o. В нём содержится машинный код программы и данные, в соответствии с исходным текстом программы, но идентификаторы пока не привязаны к адресам памяти. Для этого можно использовать один из доступных ассемблеров, например NASM. Пример команды: nasm -f format filename -o output

Второй этап — компоновка (линковка). На этом этапе при помощи программы линковщика (На Linux это ld, который есть в любой версии этой ОС) происходит связь кода из объектного файла со специальным системным кодом запуска программ, всем идентификаторам даются окончательные адреса памяти или обеспечивается их динамическое выделение. В результате создаётся исполняемый файл. Пример: ld -m elf_i386 -o output_executable input.o

Задача 5

Перепишите программу в ваш отчет, и заполните комментариями о содержании регистров все строки в сегменте кода.

Ответ

Будем комментировать только строки, в которых содержимое какого-либо из регистров меняется. Будем показывать только содержимое регистров, значение которых измениось в результате выполнения текущей команды. Все остальные регистры остаются неизменными. Предполагаем что сегмент data имеет нулевой адрес (000h)

```
; Даны два массива ArrayA, ArrayB, состоящие из 10 элементов каждый; Сравнить эти
↪  массивы поэлементно и - если элементы равны, то записать в соответствующий
↪  элемент третьего массива (Difference) 'Y' - иначе 'N' Найти сумму и количество
↪  всех одинаковых и различных элементов двух массивов (ArrayA,ArrayB)
.model tiny
.stack 100h
.data
ArrayA db 05,10,06,44,20,32,05,11,46,0
ArrayB db 35,10,15,44,20,02,65,10,46,0
Difference db 10 dup(0)
NumOfDiff dw 0
NumOfEqual dw 0
.code
start:
mov ax,@data ; ax ArrayA - 000h в регистре адрес начала массива
mov ds,ax ; ds (data segment) ArrayA - 000h адрес начала массива
push ds ; ArrayA , SP: 000h помещаем адрес на стек
pop es ; ArrayA ES, ES: 000h извлекаем адрес из стека и помещаем в
mov di,offset Difference ; DI: 014h
mov cx,10 ; CX: 10
mov al,'Y' ; al : 'Y'
cld ; Clears the DF flag in the EFLAGS register - DF: 0
rep stosb ; store contents of AL into memory address pointed by ES:DI
mov si,offset ArrayA ; si: 000h
mov di,offset ArrayB ; di: 00Ah
mov bx,offset Difference ; bx: 014h
mov cx,10 ; cx: 10 — loop count (used by loop instruction)
cld ; DF: 0
findDE:
```

```
cmpsb ; ECX: 1, SI: 001h, di: 00Bh (compares DS:SI to ES:DI)
; second iteration: SI: 002h, di: 00Ch
jne NotEqual ; IP: [NotEqual]
;second iteration: IP: [IP +1]
inc NumOfEqual
inc bx ; bx: 016h
dec di ; di: 00Bh
dec si ; si: 001h
mov al,byte ptr ds:[si] ; al: 00Ah ( 1010 )
cbw ;ah: 00Ah — sign extension AL AH (value and sign remain same) →
add SumOfEqual, ax
mov al,byte ptr ds:[di] ; al: 00Ah ( 1010 )
cbw ;ah: 00Ah — sign extension AL AH (value and sign remain same) →
add SumOfEqual, ax
inc si ; si: 002h
inc di ; di: 00Ch
jmp NextElement ; IP: [NextElement]
NotEqual:
inc NumOfDiff
mov byte ptr ds:[bx],'N'
inc bx ; bx: 015h
dec di ; di: 00Ah
dec si ; si: 000h
mov al,byte ptr ds:[si] ; al: 005h ( 510 )
cbw ; ah: 05 — sign extension AL AH (value and sign remain same) →
add SumOfDiff, ax
mov al,byte ptr ds:[di] ; al: 023h ( 3510 )
cbw ; ah: 023h
add SumOfDiff, ax
inc si ; si: 001h
inc di ; di: 00Bh
NextElement:
loop findDE ; cx: 9
mov ax,4c00h ; ax: 4c00h
int 21h
end start
```

Задача 6

В исходной строке указанное число символов, начиная с заданной позиции, переписать в конец строки.

Ответ

```
SYS_EXIT equ 1
SYS_WRITE equ 4
STDOUT equ 1
index equ 4 ;позиция 4
number equ 3 ;число символов 3
section .text
 global _start ;must be declared for linker (ld)
_start: ;tell linker entry point
;position source at index, destination at buffer
mov ESI, string + index ; SI - source register for MOVSB
mov EDI, buffer ; DI - destination register for MOVSB
mov ECX, number ; CX is used in MOVSB as a counter
REP MOVSB ;mov n bytes
;position source at index + n, destination at index
```

```
MOV EDI, string + index
MOV ESI, string + index + number
MOV ECX, number
REP MOVSB ;mov the ending of the string to the index position
MOV EDI, string + index + number ;set destination at index + n;
MOV ESI, buffer
MOV ECX, number
REP MOVSB
mov edx, length ; message length
mov ecx, string ; message to write
mov ebx, 1 ; file descriptor (stdout)
mov eax, SYS_WRITE ; system call number (sys_write)
int 0x80 ; call kernel
mov eax, SYS_EXIT ; system call number (sys_exit)
int 0x80 ; call kernel
section .data
string db '0123456789' ;Исходная строка '0123456789'
length equ $ - string
buffer times number db 0
```

Выводы

Знание языка ассемблера облегчает понимание архитектуры компьютера и работы его аппаратной части, то, чего не может дать знание языков высокого уровня абстракции ЯВУ). Для успешного использования ассемблера необходимы сразу три вещи: • Знание синтаксиса транслятора ассемблера, который используется; • Понимание машинных инструкций, выполняемых процессором во время работы • программы; • Умение работать с сервисами, предоставляемыми операционной системой

Список источников

1. https://habr.com/post/345748/ - Как писать на ассемблере в 2018 году.
2. http://rus-linux.net/MyLDP/algol/get_started_with_assembly_language_1.html - Преодолеваем ограничения высокоуровневых языков программирования и разбираемся, как на самом деле работает центральный процессор
3. https://www.tutorialspoint.com/assembly_programming/index.htm - Assembly Programming Tutorial
4. https://stackoverflow.com/questions/tagged/assembly — community answered programming question tagged with keyword «assembly»