МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Ассемблеры и их применение**

**Отчет по выполнению лабораторной работы №1**

**«Основы технологии программирования на ассемблере»**

Выполнил:

студент группы ИВТАПбд-31

Гаязов Т.Р.

Проверила:

преподаватель

Еремеева Ю.Д.

Ульяновск

2024

**Цель работы**

Приобрести навыки выполнения основных технологических операций, используемых при программировании на языке ассемблер. Написать программу, которая позволяет высчитывать булевые функции основываясь на введённых пользователем двоичных числах.

**Ход работы**

**Циклический процесс «Редактирование-ассемблирование-компоновка-выполнение»**

Данный циклический процесс служит для упрощения процесса написания-отладки при разработке на языке ассемблера.

В начале происходит создание/открытие файла с одним из следующих расширений: .asm, .mac, .inc. Затем идёт написание/редактирование кода.

После этого, с помощью компилятора (для каждого диалекта ассемблера свой) происходит ассемблирование кода -> генерация объектных файлов (.obj) из исходных, фактически текстовых файлов. В случае обнаружения ошибки компиляции требуется вернуться к редактированию исходных файлов.

Далее, по окончанию ассемблирования объектные модули с помощью программы компоновщика (для TASM ассемблера - TLINK) собираются в исполняемый файл с расширением .exe. В случае ошибки компоновки требуется вернуться к этапу редактирования.

Производится запуск исполняемого файла и проверка работы кода, при наличии ошибок требуется вернуться к редактированию и процесс повторяется до исключения всех ошибок или наращивания нужного функционала программы.

**Исходные файлы**

Исходный архив с файлами включает в себя 4 файла mlab1.asm(содержит основной код программы), mlab1l.asm (содержит в себе вспомогательные функции для ввода/вывода информации, замера времени и тд.), mlab1.inc(содержит константы, общие для всей программы) mlab1.mac(содержит в себе все макросы, используемые в программе).

**Процесс ассемблирования и компоновки**

В ходе макрообработки препроцессор компилятора TASM подставляет текст макрокоманд в место из вызова, при необходимости заменяя макроаргументы на указанные пользователем, в исходном коде программы.

При ассемблировании можно указать генерацию дополнительных файлов таких как файл перекрёстных ссылок и файл листинга.

Файл перекрестных ссылок включает в себя имена всех переменных, строки, в которых они объявлены и номера строк в которых данные переменные используются. Данный файл исключительно полезен при отладке больших программ, когда требуется просмотреть, в каких местах конкретная переменная используется.

В верхней части каждой страницы листинга выводится заголовок, содержащий версию Турбо Ассемблера, с помощью которой выполнено ассемблирование файла, дату и время ассемблирования, и номер страницы в листинге.

Файл листинга содержит две части: аннотированный листинг исходного кода и таблицу идентификаторов. Сначала выводится исходный ассемблируемый код с заголовком, содержащим имя файла, в котором находится исходный код. Вывод исходного кода Ассемблера сопровождается информацией о машинном коде, который ассемблирует

Все ошибки или предупреждения, обнаруженные в процессе ассемблирования, включаются в листинг непосредственно за содержащей ошибку строкой.

После этого происходит ассемблирование исходного файла, все команды переводятся в байт код и получается файл трансляции в формате .obj.

Компоновщик TLINK объединяет единицы трансляции и добавляет к ним код динамических библиотек в формате .lib. После этого происходит создание двоичного исполняемого файла в формате .com или .exe.

**Выполнение работы**

Циклический процесс “редактирование-ассемблирование-компоновка-выполнение” состоит из следующих этапов.

Изначально берутся файлы-исходники с кодом, после чего создается bat-файл, содержащий набор команд, предназначенных для исполнения командным интерпретатором. Циклический Bat-файл содержит следующие строки:

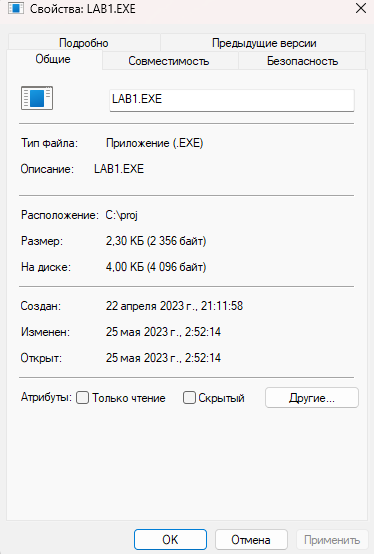
**Листинг 1. Bat-файл**

|  |
| --- |
| **tasm /la/zi mlab1.asm; mlab1l.asm**  **pause**  **tlink /v mlab1.obj mlab1l.obj, res**  **pause**  **res.exe** |

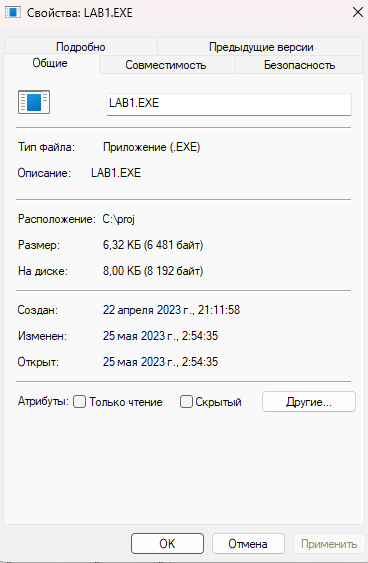
С помощью утилиты TASM ассемблируются файлы mlab1.asm и mlab1l.asm, находящиеся в папке, после чего с помощью утилиты TLINK происходит процесс компоновки этих файлов. В случае ошибок происходит вывод соответствующего сообщения. Таким образом реализуется циклический процесс «редактирование – ассемблирование – компоновка – выполнение».

Сравним размеры .exe файлов при ассемблировании с использованием параметров (zi – информация об идентификаторах для отладки - полная, la – генерация листинга - расширенная) и при компоновке с ключами m и v.

Размер файла (КБ) mlab1.exe без ключей (Рис.2):

*Рис.2 lab1.exe без ключей*

Размер файла (КБ) mlab1.exe с ключами /zi для TASM, /v для TLINK (Рис.3):

*Рис.3 lab1.exe с ключами*

Видно, что при использовании ключей размеры .exe файлов увеличиваются.

Основной код программы храниться в файле mlab1.asm. Для считывания 20 разрядных чисел использовалась метод ввода по-смовольно (досрочная терминация символом «/»). Функция считалась следующим образом: происходило обращение к определенному элементу строки, затем отнимая от полученного значения 30h получалось число, находящееся в данном разряде. Затем с помощью логических операторов сдвига были поулчены нужные биты, требуемые для сравнения (реализовано через прыжки по меткам). Умножение исходных чисел проводилось с использованием сдвига всех битов операнда влево (команда shl), а деление с использованием сдвига вправо (команда shr). Изменение разрядов в числе Z проводилось с помощью установки определенного бита в 1 или 0 (команды BTS и BTR соответственно).

**Выполнение индивидуального задания.**

В качестве индивидуального задания программа была модернизирована для ввода двух двоичных значений X, Y и вычисления на их основе двух логических функций.

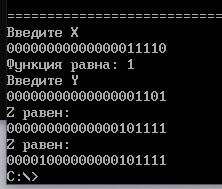
Z = f6? X ∗ 2 − Y : X/8 + Y ; z7& = z4; z9 |= z11; z15 = ¯z17;

f6 = ¯x1 x¯2 x3 | x1 x3 | x2 x¯3 | x2 x4 | x1 x¯3 x¯4.

**Тестирование**

Пример 1.

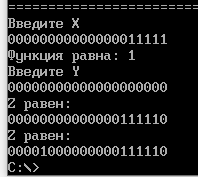
Вводим X:00000000000000011110, функция должна быть равна 1, вводим Y: 00000000000000001101 получаем Z-ки. См. рисунок 4



*Рис.4 Тест №1*

Пример 2.

Вводим X: 00000000000000011111, функция должна быть равна 1, вводим Y:0000000000000000000 получаем Z-ки. См. рисунок 5



*Рис.5 Тест №2*

**Вывод**

В ходе выполнения лабораторной работы были изучены основы работы с компиляторами, компоновщиками и отладчиками. Испытан на практике циклический процесс «редактирование-ассемблирование-компоновка-выполнение» и разработан для его автоматизации скриптовый файл bat.

**Приложение 1. Mlab1**

|  |
| --- |
| ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ; MLAB1.ASM - учебный пример для выполнения  ; лабораторной работы N1 по машинно-ориентированному программированию  ; 10.09.02: Негода В.Н.  ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .MODEL SMALL  .STACK 200h  .386  ; Используются декларации констант и макросов  INCLUDE MLAB1.INC  INCLUDE MLAB1.MAC  ; Декларации данных  .DATA  SLINE DB 78 DUP (CHSEP), 0  REQ DB "Фамилия И.О.: ",0FFh  MINIS DB "МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ",0  ULSTU DB "УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ",0  DEPT DB "Кафедра вычислительной техники",0  MOP DB "Машинно-ориентированное программирование",0  LABR DB "Лабораторная работа N 1",0  REQ1 DB "Замедлить(-),ускорить(+), посчитать функцию f(f), выйти(ESC)? ",0FFh  TACTS DB "Время работы в тактах: ",0FFh  EMPTYS DB 0  BUFLEN = 70  BUF DB BUFLEN  LENS DB ?  SNAME DB BUFLEN DUP (0)  PAUSE DW 0, 0 ; младшее и старшее слова задержки при выводе строки  TI DB LENNUM+LENNUM/2 DUP(?), 0 ; строка вывода числа тактов  LAB1\_MSG\_IN\_X DB "Введите X", 0  LAB1\_MSG\_IN\_Y DB "Введите Y", 0  LAB1\_F\_VAR DB "f3 = x2x4 | x1!x3 | !x1x3!x4 | x3x4 | !x1!x2x3", 0  LAB1\_MSG\_F DB "Функция равна: $"  LAB1\_MSG\_Z DB "Z равен:", 0  X\_ DB 20 dup(0)  X DD 0  Y DD 0  Z DD 0  F DD 0  ; запас для разделительных "`"  ;========================= Программа =========================  .CODE  ; Макрос заполнения строки LINE от позиции POS содержимым CNT объектов,  ; адресуемых адресом ADR при ширине поля вывода WFLD  BEGIN LABEL NEAR  ; инициализация сегментного регистра  MOV AX, @DATA  MOV DS, AX  ; инициализация задержки  MOV PAUSE, PAUSE\_L  MOV PAUSE+2,PAUSE\_H  PUTLS REQ ; запрос имени  ; ввод имени  LEA DX, BUF  CALL GETS  @@L: ; циклический процесс повторения вывода заставки  ; вывод заставки  ; ИЗМЕРЕНИЕ ВРЕМЕНИ НАЧАТЬ ЗДЕСЬ  FIXTIME  PUTL EMPTYS  PUTL SLINE ; разделительная черта  PUTL EMPTYS  PUTLSC MINIS ; первая  PUTL EMPTYS  PUTLSC ULSTU ; и  PUTL EMPTYS  PUTLSC DEPT ; последующие  PUTL EMPTYS  PUTLSC MOP ; строки  PUTL EMPTYS  PUTLSC LABR ; заставки  PUTL EMPTYS  ; приветствие  PUTLSC SNAME ; ФИО студента  PUTL EMPTYS  ; разделительная черта  PUTL SLINE  ; ИЗМЕРЕНИЕ ВРЕМЕНИ ЗАКОНЧИТЬ ЗДЕСЬ  DURAT ; подсчет затраченного времени  ; Преобразование числа тиков в строку и вывод  LEA DI, TI  CALL UTOA10  PUTL TACTS  PUTL TI ; вывод числа тактов  ; обработка команды  PUTL REQ1  CALL GETCH  CMP AL, '-' ; удлиннять задержку?  JNE CMINUS  INC PAUSE+2 ; добавить 65536 мкс  JMP @@L  CMINUS: CMP AL, '+' ; укорачивать задержку?  JNE LAB1\_CFUNC  CMP WORD PTR PAUSE+2, 0  JE BACK  DEC PAUSE+2 ; убавить 65536 мкс  BACK: JMP @@L  LAB1\_CFUNC:  CMP AL, 'f'  JE LAB1\_CODE  CMP AL, 'F'  JE LAB1\_CODE  JMP CEXIT  LAB1\_CODE:  MOV ECX, 5  PUTL EMPTYS  PUTL EMPTYS  PUTL EMPTYS  PUTL EMPTYS  PUTL EMPTYS  PUTL SLINE    PUTL LAB1\_MSG\_IN\_X  MOV ECX, 20  LEA ESI, X\_ + 20; в ESI хранится адрес конца массива  XOR EDX,EDX  LAB1\_IN\_X:  SHL EDX, 1  DEC SI  BTR DWORD PTR [SI], 0  CALL GETCH  CALL PUTC  CMP AL, '1'  JNE LAB1\_IN\_X\_NZ  ADD DWORD PTR [SI], 1  ADD EDX, 1  LAB1\_IN\_X\_NZ:  LOOP LAB1\_IN\_X  MOV X, EDX  MOV AL, 10  CALL PUTC  MOV AL, 13  CALL PUTC  ;теперь x1: [ESI + 1] и т.д.  LAB1\_FUNC\_CALC:  ;f6 = ?x1 x?2 x3 | x1 x3 | x2 x?3 | x2 x4 | x1 x?3 x?4.  XOR EAX, EAX  XOR EBX, EBX  MOV AL, [ESI+1] ; x1  MOV BL, [ESI+2] ; x2  BTC AX, 0  BTC BX, 0  AND AL, BL ; !x1!x2  MOV BL, [ESI+3] ; x3  AND AL, BL ; !x1!x2x3  MOV F, EAX ; X = !x1!x2x3  MOV AL, [ESI+1] ; x1  MOV BL, [ESI+3] ; x3  AND AL, BL ; x1x3  OR EAX, F ; !x1!x2x3 | x1x3  MOV F, EAX ; X = !x1!x2x3 | x1x3  MOV AL, [ESI+2] ; x2  MOV BL, [ESI+3] ; x3  BTC BX,0 ; !x3  AND AL, BL ; x2!x3  OR EAX, F ; !x1!x2x3 | x1x3| x2!x3  MOV F, EAX ; X = !x1!x2x3 | x1x3 | x2!x3  MOV AL, [ESI+2] ; x2  MOV BL, [ESI+4] ; x4  AND AL, BL ; x2x4  OR EAX, F ; !x1!x2x3 | x1x3 | x2!x3 | x2x4  MOV F, EAX ; X = !x1!x2x3 | x1x3 | x2!x3 | x2x4  MOV AL, [ESI+1] ; x1  MOV BL, [ESI+3] ; x3  BTC BX,0 ; !x3  AND AL, BL ; x1!x3  MOV BL, [ESI+4] ; x4  BTC BX,0 ; !x4  AND AL, BL ; x1!x3!x4  OR EAX, F ; !x1!x2x3 | x1x3 | x2!x3 | x2x4 | x1!x3!x4  MOV F, EAX ; X = !x1!x2x3 | x1x3 | x2!x3 | x2x4 | x1!x3!x4  MOV AH, 09h  LEA DX, LAB1\_MSG\_F  INT 21h  ADD AX, '0'  CALL PUTC  PUTL EMPTYS  PUTL LAB1\_MSG\_IN\_Y  MOV ECX, 20 ; Ввод Y  XOR EDX, EDX  LAB1\_IN\_Y:  SHL EDX, 1  CALL GETCH  CALL PUTC  CMP AL, '1'  JNE LAB1\_IN\_Y\_NZ  ADD EDX, 1  LAB1\_IN\_Y\_NZ:  LOOP LAB1\_IN\_Y    MOV Y, EDX  PUTL EMPTYS  CMP F, 0 ; Проверка функции  JE LAB1\_F\_FALSE  JNE LAB1\_F\_TRUE  LAB1\_F\_TRUE:  SHL X, 1 ; X \* 2  MOV EAX, X  SUB EAX, Y  MOV Z, EAX ; Z = X\*2 - Y  JMP LAB1\_Z\_FIRST  LAB1\_F\_FALSE:  SHR X, 3 ; X / 8  MOV EAX, X  ADD EAX, Y  MOV Z, EAX ; Z = X/8 + Y  LAB1\_Z\_FIRST:  MOV EBX, Z  XOR EAX, EAX  PUTL LAB1\_MSG\_Z  SHL EBX, 12 ; сдвиг лишних 12 битов слева  MOV ECX, 20  LAB1\_OUT\_Z\_0:  SHL EBX, 1 ; Сдвиг значения EBX на 1 бит влево  DEC ECX ; Уменьшение счетчика  JC LAB1\_OUT\_Z\_NZ\_0  LAB1\_OUT\_Z\_Z\_0:  MOV AL, '0' ; Если бит был 0, выводим '0'  CALL PUTC  JMP CHECK\_EXIT\_0  LAB1\_OUT\_Z\_NZ\_0:  MOV AL, '1' ; Если бит был 1, выводим '1'  CALL PUTC  CHECK\_EXIT\_0:  CMP ECX, 0 ; Проверяем, закончили ли мы вывод всех бит  JG LAB1\_OUT\_Z\_0 ; Если да, то продолжаем вывод  MOV AL, 10  CALL PUTC  MOV AL, 13  CALL PUTC  MOV EAX, Z  BT EAX, 11 ; проверка 11-го бита  JC LAB1\_Z\_FIRST\_NZ  MOV EBX, 0 ; если 11-й бит равен 0  JMP LAB1\_Z\_FIRST\_CALC  LAB1\_Z\_FIRST\_NZ:  MOV EBX, 1 ; если 11-й бит равен 1  LAB1\_Z\_FIRST\_CALC:  SHL EBX, 9 ; создание маски для операции OR  MOV EAX, Z  OR EAX, EBX ; применение маски  MOV Z, EAX  BT EAX, 17 ; проверка 17-го бита  JC LAB1\_Z\_SECOND\_NZ  BTS EAX, 15 ; если 17-й бит равен 0, то 18-й бит равен 1  JMP LAB1\_Z\_THIRD  LAB1\_Z\_SECOND\_NZ:  BTR EAX, 15 ; если 17-й бит равен 1, то 18-й бит равен 0  LAB1\_Z\_THIRD:  MOV Z, EAX  BT EAX, 4 ; проверка 4-го бита  JC LAB1\_Z\_THIRD\_END ; если 4-й бир равен 1, то 7-й бит не изменяем  LAB1\_Z\_THIRD\_Z:  BTR EAX, 7 ; если 4-й бит равен 0, то 7-й бит равен 0  LAB1\_Z\_THIRD\_END:  MOV Z, EAX    PUTL LAB1\_MSG\_Z  MOV EBX, Z  XOR EAX, EAX  SHL EBX, 12 ; сдвиг лишних 12 битов слева  MOV ECX, 20  LAB1\_OUT\_Z:  SHL EBX, 1 ; Сдвиг значения EBX на 1 бит влево  DEC ECX ; Уменьшение счетчика  JC LAB1\_OUT\_Z\_NZ  LAB1\_OUT\_Z\_Z:  MOV AL, '0' ; Если бит был 0, выводим '0'  CALL PUTC  JMP CHECK\_EXIT  LAB1\_OUT\_Z\_NZ:  MOV AL, '1' ; Если бит был 1, выводим '1'  CALL PUTC  CHECK\_EXIT:  CMP ECX, 0 ; Проверяем, закончили ли мы вывод всех бит  JG LAB1\_OUT\_Z ; Если да, то продолжаем вывод  JLE @@E ; Если нет, то выходим  CEXIT: CMP AL, CHESC  JE @@E  TEST AL, AL  JNE BACK  CALL GETCH  JMP @@L  ; Выход из программы  @@E: EXIT  EXTRN PUTSS: NEAR  EXTRN PUTC: NEAR  EXTRN GETCH: NEAR  EXTRN GETS: NEAR  EXTRN SLEN: NEAR  EXTRN UTOA10: NEAR  END BEGIN |