ПРЕДИСЛОВИЕ

Микропроцессор (МП) Intel x86 – 32-разрядный однокристальный центральный процессор с фиксированным набором команд. На основе МП Intel x86 выпускаются различные типы вычислительных систем, включая широко распространенные IBM-совместимые персональные компьютеры, которые имеют развитое программное обеспечение, в том числе трансляторы с языков высокого уровня и среды визуального программирования. В настоящем издании рассматривается программирование персонального компьютера на языке низкого уровня – языке ассемблера, несмотря на трудоемкость и меньшее удобство его использования. В каких случаях необходимо использование языка ассемблера?

Во-первых, для создания программ с максимальной производительностью, например, в сложных игровых программах, машинной графике и т.п. Во-вторых, для написания интерфейсных программ, обеспечивающих взаимодействие между языками высокого уровня и некоторыми служебными процедурами операционной системы. Наконец, для самостоятельной доработки готовых программ. Таким образом, изучение программирования персонального компьютера на языке ассемблера является необходимым элементом подготовки будущего специалиста по вычислительной технике.

Разделы настоящего издания (этапы разработки программ, использование макросредств, организация внешних подпрограмм, обработка сложных структур данных, программирование блока с плавающей точкой) последовательно, от простого к сложному, знакомят студента с технологией разработки программ на языке ассемблера.

Лабораторные работы выполняются на персональном компьютере, совместимом с IBM PC, в виртуальной DOS-машине Windows. Программирование защищенного режима процессоров x86 и пользовательское программирование в операционной системе Windows рассматривается в последующих частях практикума.

Лабораторная работа 1. ПОДГОТОВКА, ТРАНСЛЯЦИЯ И ОТЛАДКА ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА

Разработка программы на языке ассемблера складывается из следующих этапов:

– постановка задачи, разработка алгоритма и текста программы; набор текста программы на ЭВМ с помощью редактора текстов;

– трансляция программы – получение объектного файла с помощью программы ассемблера (TASM – Turbo Assembler);

– получение загрузочного файла с помощью редактора связей (TLINK – Turbo Linker);

– отладка программы с помощью отладчика (TD – Turbo Debugger);

– исполнение программы и проверка результатов.

Исходная программа представляет собой последовательность операторов. Оператором может быть либо машинная команда микропроцессора (прил. 1, 2), либо псевдокоманда языка ассемблера (прил. 3). Операторы любого типа могут включать в себя операции, которые дают ассемблеру информацию об операндах (прил. 4).

Рассмотрим пример разработки и отладки программы.

Задание: Написать программу сложения двух одноразрядных десятичных чисел, вводимых с клавиатуры.

Решение:

1. Текст программы набирается с помощью ASCII-редактора. Имя программы произвольное, расширение ASM (например, demo.asm).

```
;Демонстрационная программа сложения двух
;одноразрядных беззнаковых чисел
;Для построения рабочей версии используйте
;команды:
; >tasm demo;
; >tlink demo;
; >demo
; Резервирование места под стек
sseg segment stack 'stack'
     dw 256 dup(?)
sseg ends
```

```
; Определение данных
data segment
; Сообщения пользователю
msg1 db 10,13,'Программа сложения двух чисел'
     db 10,13,'Введите первое число: ','$'
msg2 db 10,13,'Введите второе число: ','$'
msg3 db 10,13,'Результат = ','$'
data ends
; Сегмент кода
code segment
assume cs:code,ds:data,ss:sseg
start: mov ax,data ; настроить сегментный
       mov ds,ax   ; регистр DS на данные
       lea dx,msg1 ; вывести сообщение
       call print_msg
       call input_digit; ввести первое число
       mov bl,al    ; и сохранить в регистре BL
       lea dx,msg2  ; вывести сообщение
       call print_msg
       call input_digit ; ввести второе число
       lea dx,msg3      ; вывести сообщение
       call print_msg
       call add_and_show; сложить и вывести
                        ;  результат
       mov ah,4ch       ; завершить программу
       int 21h             ; и выйти в DOS
; Подпрограмма вывода сообщения на дисплей
; Вход: DS:DX - адрес сообщения
; Выход: вывод сообщения на дисплей
print_msg proc
       push ax     ; сохранить AX
       mov ah,09h ; вывести сообщение
       int 21h    ;  с помощью функции DOS
       pop ax     ; восстановить AX
       ret  ; вернуться в вызывающую программу
print_msg endp
; Подпрограмма ввода числа с клавиатуры
; Вход: набранная с клавиатуры цифра
; Выход: в AL - введенное число
input_digit proc
input_again:
       mov ah,01h  ; ввести символ с клавиатуры
```

```
        int 21h     ;  с помощью функции DOS
        cmp al,'0'  ; если символ не цифра,
        jl input_again ; то повторить ввод
        cmp al,'9'
        jg input_again
        sub al,30h; преобразовать код символа
                  ;  в число
        ret  ; вернуться в вызывающую программу
input_digit endp
; Подпрограмма сложения двух чисел
;Вход: AL,BL – слагаемые,
;выход: вывод результата на дисплей
add_and_show proc
        add al,bl   ; сложить (AL=AL+BL)
        cmp al,9    ; если результат > 9,
        jle not carry  ;  то уменьшить сумму на
        sub al,10      ;  10 и вывести на дисплей
        push ax        ;  символ '1' – старшую
        mov ah,2h      ;  цифру результата
        mov dl,'1'     ;  с помощью функции DOS
        int 21h
        pop ax
not_carry: add al,30h ; преобразовать число
                      ;  в код символа
        mov ah,2h     ; вывести младшую цифру
        mov dl,al     ;  результата с помощью
        int 21h       ;  функции DOS
        ret  ; вернуться в вызывающую программу
add_and_show endp
code ends
end start
```

Программа состоит из трех логических сегментов: стека SSEG, данных DATA, кода CODE. Каждый сегмент начинается с псевдокоманды SEGMENT и заканчивается псевдокомандой ENDS, причем обе псевдокоманды для одного и того же сегмента имеют одинаковые имена.

Сегмент стека содержит псевдокоманду DW с операндом 256 DUP (?), что означает зарезервировать для стека 256 слов памяти, но не инициализировать их.

Сегмент данных включает в себя сообщения пользователю, выдаваемые в процессе выполнения программы. Коды 10 и 13 задают управляющие символы возврата каретки и перевода строки. Сообщения завершаются символом $ в соответствии с требованиями функции DOS, с помощью которой они выводятся.

Сегмент кода начинается с псевдокоманды ASSUME, которая сообщает ассемблеру, что регистр CS будет содержать базовый адрес сегмента CODE, регистр DS – сегмента DATA, регистр SS – сегмента SSEG. Сегментный регистр ES в программе не используется, поэтому в псевдокоманде ASSUME не указывается, что по умолчанию соответствует значению NOTHING (ничего). Сегментные регистры CS и SS, указатель стека SP загружает операционная система. Загрузка сегментных регистров DS и ES возлагается на программиста, поэтому в первых двух командах сегмента кода производится инициализация используемого регистра DS.

Далее следует собственно программа, соответствующая заданию. Она оформлена в виде основной программы и вызываемых в ней подпрограмм. Последняя строка программы содержит псевдокоманду конца трансляции END с операндом START, указывающим первую исполняемую команду программы.

2. Для ассемблирования программы необходимо набрать

>TASM имя

Ассемблеру дается указание обработать исходный файл имя.ASM для создания объектного файла имя.OBJ и файла листинга имя.LST. Если ассемблер выдал сообщения об ошибках, то необходимо исправить исходную программу с помощью редактора текстов и заново ее оттранслировать.

3. Для редактирования связей вызывается компоновщик

>TLINK имя

Сформированный загрузочный файл имеет расширение EXE (например, DEMO.EXE).

4. Отладку программы можно выполнять только в том случае, если ассемблер не обнаружил ошибок в исходном тексте, а редактор связей создал файл типа EXE. Загрузочный файл можно вызвать из операционной системы или из отладчика. Программа вызывается из OS только тогда, когда есть уверенность в ее пра-

вильности или когда она выдает какие-либо видимые результаты. Будем использовать отладчик Turbo Debugger.

Turbo Debugger (TD) – мощный отладчик, используемый при программировании на языках Turbo C++, Turbo Pascal и Turbo Assembler. TD позволяет вести отладку как на уровне машинного кода, так и на уровне исходного текста программы.

Пусть отлаживаемая программа находится в файле demo.exe. Тогда TD запускается следующим образом:

>TD demo.exe

Можно также запустить TD без параметров, затем нажать F3 и выбрать имя нужного файла из списка. Выход из отладчика – Alt-X.

Вся информация в отладчике представляется как ряд окон (табл. 1). Окно имеет название (например, CPU - окно центрального процессора) и номер (верхний правый угол рамки).

Таблица 1

Команды работы с окнами

Сочетание клавиш	Команда
ALT-F3	Закрыть активное окно
F5	Распахнуть окно на весь экран, повторное нажатие вернет окну прежние размеры
CTRL-F5	Перейти в режим перемещения окна, 'стрелки' перемещают окно, а при нажатой клавише Shift – меняют его размер. Выход из режима по нажатию Enter
F6	Перейти к следующему окну
ALT-F5	Переключиться на экран отлаживаемой программы
ALT-цифра 1:9	Перейти к окну с соответствующим номером.
Tab	Перейти к другой панели активного окна (некоторые окна состоят из нескольких частей – 'панелей', например, окно CPU содержит панели дизассемблированного кода, регистров и дампа)
ALT-V C	Открыть окно CPU (если оно еще не открыто)
ALT-V N	Открыть окно сопроцессора, в нем отображается содержимое всех регистров стека сопроцессора и его флаги.
ALT-V D	Открыть окно дампа памяти

Для того чтобы изменить содержимое какого-либо регистра ЦП, необходимо:

1) сделать окно CPU активным (ALT-1);

2) перейти в панель регистров (Tab);

3) выбрать нужный регистр (стрелками);

4) набрать новое значение (например, 1200 'ENTER').

Аналогично можно менять содержимое участков памяти (в окне DUMP или в нижней панели окна CPU) и содержимое регистров стека сопроцессора (в окне Numeric Coprocessor).

Команды выполнения программы приведены в табл. 2.

Таблица 2

Команды отладки

Клавиша	Команда
F7	Выполнить одну команду
F4	Выполнять команды до той, на которую установлен курсор (в окне CPU)
F2	Снять/поставить точку останова (команда, на которой стоит точка останова, выделяется цветной полоской)
F9	(RUN) запустить программу на выполнение. Программа остановится, если достигнута точка останова или конец программы (mov ah, 4ch/int 21h)
F1	Подсказка

5. Исполнение программы**.** Если программа отлажена, то ее можно запускать без отладчика:

>имя

Задание к лабораторной работе

1. Оттранслировать текст программы demo.asm, построить exe-файл и запустить программу на выполнение.

2. Изучить работу программы в отладчике.

3. Изменить текст процедуры ADD_AND_SHOW так, чтобы производилось не сложение, а вычитание (SUB_AND_SHOW) двух одноразрядных целых десятичных чисел, причем в результате получалось одноразрядное положительное или отрицательное число (например, 8 – 3 = 5, 3 – 8 = –5). Использовать команды SUB, JNS, NEG.

4. Продемонстрировать работу измененной программы преподавателю.

Содержание отчета

1. Цель работы.

2. Текст процедуры SUB_AND_SHOW и блок-схема алгоритма.

3. Описание используемых функций DOS.

Контрольные вопросы

1. Какие системные программы используются при разработке и отладке программ на языке ассемблера?

2. Сколько байтов памяти зарезервирует следующая последовательность операторов:

```
X1      DW      5
X2      DB      3 DUP(?), 7
X3      DW      6 DUP(?)
```

3. Что такое физический и логический адреса в микропроцессорах 80x86?

4. Какие регистры микропроцессора могут участвовать в формировании физического адреса при выборке команд, при обращении к переменным?

5. Какие три атрибута имеет переменная?

6. Какие ошибки имеются в каждом из приведенных ниже фрагментов программ:

```
а) CONST EOU   256
              ...
  MOV  CONST, AX
б) X   DB ?
       ...
       ...
   MOV X, AX
в) X   DB ?
  Y   DB 25
        ...
   MOV   X, Y
г) MOV [BX][BP], AX
```

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Брэй Б. Микропроцессоры Intel: 8086/8088…80486, Pentium: пер. с англ. СПб: BHV-Петербург, 2005. 1328 с
2. Зубков С.В. Ассемблер для DOS, Windows и UNIX. М.: ДМК-Пресс, 2013. 638 с.
3. Таненбаум Э. Архитектура компьютера: 5-е изд. пер. с англ. СПб.: Питер, 2013. 848 с.
4. Юров В.И. Assembler: учебник для вузов. СПб.: Питер, 2011. 640 с.
5. Юров В.И. Assembler: практикум: учеб. Пособие для вузов. СПб.: Питер, 2007. 400 с.

Приложения

1. Основные псевдокоманды ассемблера

Псевдокоманда	Назначение
Управление сегментированием и счетчиком команд	
SEGMENT	Формат: имя-сег SEGMENT [тип-выравнивания] [использование][тип-объединения] ['имя_класса'] ... имя-сег ENDS Определяет границы сегмента программы
ASSUME	Формат: ASSUME.сег_регистр: имя_сег[,...] или ASSUME сег_регистр: NOTHING Сообщает ассемблеру, какой регистр сегмента связан с сегментом программы. Оператор ASSUME NOTHING отменяет действие всех предыдущих операторов ASSUME для данного регистра
ORG	Формат: ORG выражение Устанавливает счетчик адреса, равным значению выражения. Ассемблер присвоит этот адрес следующему объектному коду
EVEN	Формат: EVEN Сдвигает значение счетчика адреса к ближайшему четному байту

Псевдокоманда	Назначение
GROUP	Формат: имя-группы GROUP имя_сег[,...] Определяет список объектов, которые должны быть размещены в одном блоке памяти объемом 64 Кбайтов
LABEL	Формат: имя LABEL тип для любых ячеек памяти. Если указан тип BYTE, WORD и DWORD, то имя представляет собой переменную. Имя с типом NEAR или FAR представляет собой метку
Определение данных (переменных)	
DB	Формат: [имя] DB выражение [,...] Определяет байты памяти
DW	Формат: [имя] DW выражение [,...] Определяет два байта памяти
DD	Формат: [имя] DD выражение [,...] Определяет четыре байта памяти
DQ	Формат: [имя] DQ выражение [,...] Определяет восемь байтов памяти
DT	Формат: [имя] DT выражение [,...] Определяет десять байтов памяти
Определение символических имен	
EQU	Формат: имя FQU выражение Постоянно присваивает значение выражения имени
=	Формат: имя = выражение Значение выражения присваивается имени, но может быть переприсвоено
Определение процедуры	
PROC	Формат: имя PROC [NEAR] или имя PROC FAR ... RET имя ENDP Определяет процедуру
Связывание модулей	
PUBLIC	Формат: PUBLIC имя [,...] Объявляет имя доступным другим модулям программы, которые должны быть присоединены к данному модулю

Псевдокоманда	Назначение
EXTRN	Формат: EXTRN имя: тип [,...] Указывает, что имя определено в другом модуле программы
NAME	Формат: NAME имя_модуля Присваивает внутреннее имя объектному модулю, генерируемому ассемблером
END	Формат: END [метка точки входа] Конец ассемблирования
Управление листингом	
PAGE	Формат: PAGE [число строк][,число столбцов] Устанавливает длину и ширину печатаемой страницы. По умолчанию размер страницы 57 строк по 80 символов
TITLE SUBTITLE	Формат: TITLE текст Формат: SUBTITLE текст Указывают заголовок и подзаголовок, которые должны быть напечатаны на второй и третьей строке страницы листинга соответственно. На верхней строке каждой страницы ассемблер печатает номер главы и номер страницы, разделяя их дефисом. Ассемблер увеличивает номер главы, если встретился оператор PAGE+, при этом нумерация страниц заново начинается с 1
XLIST	Формат: .XLIST Отменяет листинг программы вплоть до появления псевдокоманды .LIST
LIST	Формат: .LIST Возобновляет листинг программы
%OUT	Формат: %OUT ТЕКСТ Выдает сообщение ТЕКСТ во время трансляции
Псевдокоманды упрощенной сегментации	
.MODEL	Формат: .model модель_памяти Определяет используемую в программе модель памяти
.STACK	Формат: .stack [выражение] Начало сегмента стека с размером, задаваемым выражением (по умолчанию 1024 байта)
.DATA	Начало сегмента данных

Псевдокоманда	Назначение
.CODE	Начало сегмента кода
Псевдокоманды разрешения ассемблирования команд заданного процессора	
.8086	Разрешает ассемблирование команд процессора 8086 и сопроцессора 8087 (по умолчанию)
.286	Разрешает ассемблирование непривилегированных команд процессора 80286 и сопроцессора 80287 (и ниже). Аналогично: .386, .486, .586, .686.
.286P	Разрешает ассемблирование всех команд процессора 80286 (включая защищенный режим) и сопроцессора 80287 (и ниже). Аналогично: .386P, .486P, .586P, .686P.

Примечание. Конструкция [ ] или [,…] обозначает, что находящиеся в квадратных скобках аргументы или могут не появляться ни одного раза, или появиться произвольное число раз.

2. Операции ассемблера

Операция	Назначение
Арифметические и логические	
+	Арифметическое сложение
-	Арифметическое вычитание
*	Арифметическое умножение
/	Целочисленное деление
MOD	Остаток от целочисленного деления
AND	Логическое И
OR	Логическое ИЛИ
XOR	Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ
NOT	Логическое НЕ
EQ	Сравнение на равенство
NE	Сравнение на не равенство
LT	Сравнение на меньше
GT	Сравнение на больше
LE	Сравнение на меньше или равно
GE	Сравнение на больше или равно
SHL	Формат: значение SHL выражение Сдвигает значение влево на число битов, равное выражению
SHR	Формат: значение SHR выражение Сдвигает значение вправо на число битов, равное выражению

Операция	Назначение
HIGN	Формат: HIGH значение или HIGH выражение Выделяет старший байт 16-битного числового значения или выражения
LOW	Формат: LOW значение или LOW выражение Выделяет младший байт 16-битного числового значения или выражения
Определение компонент адресов и переменных	
$	Формат: $ Определяет значение счетчика адреса текущей ячейки
SEG	Формат: SEG переменная или SEG метка Определяет сегментную компоненту адреса переменной или метки
OFFSET	Формат: OFFSET переменная или OFFSET метка Определяет смещение адреса переменной или метки
LENGTH	Формат: LENGTH переменная Определяет длину в единицах определения (байтах или словах) любой переменной, при определении которой был использован псевдооператор DUP
TYPE	Формат: TYPE переменная или TYPE метка Для переменной операция TYPE возвращает 1, если она имеет тип BYTE, 2 – WORD, 4 – DD, | 8 – DQ, 10 – DT. Для меток она возвращает (атрибут NEAR), 2 – атрибут FAR.
SIZE	Формат: SIZE переменная Определяет произведение LENGTH×TYPE
DUP	Формат: n DUP (выражение [,…]) Используется в псевдокомандах определения данных для распределения и инициализации нескольких (n) единиц памяти
?	Формат: ? Используется в псевдокомандах определения данных для резервирования памяти без ее начальной инициализации
Присваивание атрибута	

Операция	Назначение
PTR	Формат: тип PTR переменная или тип PTR метка или тип PTR адресное-выражение Изменяет атрибут типа (BYTE, WORD) или атрибут дистанции (NEAR, FAR) переменной, метки или адресного выражения
SHORT	Формат: SHORT адресное выражение SHORT метка Сообщает ассемблеру о том, что для представления значения выражения будет достаточно 1 байта
THIS	Формат: THIS атрибут_типа или THIS атрибут_дистанции Создает переменную или метку с заданным атрибутом, смещение и сегмент которых являются текущими значениями счетчика адреса
Тип выравнивания	
BYTE	Определяет границу, на которой должен быть размещен последующий логический сегмент при объединении сегментов Сегмент может начинаться по любому адресу
WORD	Сегмент должен начинаться с четного адреса
PARA	Сегмент должен начинаться с адреса, кратного 16.
PAGE	Сегмент должен начинаться с адреса, кратного 256.
INPAGE	Весь сегмент должен быть расположен в пределах страницы в 256 байт
Использование	
USE16	Определяет, что по умолчанию используется 16-битный размер операнда и адреса
USE32	Определяет, что по умолчанию используется 32-битный размер операнда и адреса (начиная с процессора 80386 в защищенном режиме)
Тип объединения	
PUBLIC	Определяет последовательное расположение сегментов кодов и/или данных модулей программы в памяти
STACK	Определяет последовательное расположение стековых сегментов модулей программы в памяти
COMMON	Определяет, что данный сегмент разделяет одинаковые ячейки памяти со всеми другими сегментами из других модулей, имеющими такие же имена

Операция	Назначение
MEMORY	Аналогичен COMMON, но в памяти сегменты с данным типом объединения размещаются после всех других сегментов
Модель памяти	
TINY	Определяет минимальную модель памяти (в программе используется общий сегмент кода, данных и стека)
SMALL	Определяет малую модель памяти (в программе используются отдельные сегменты кода и данных)
MEDIUM	Определяет среднюю модель памяти (в программе используются несколько сегментов кода и сегмент данных)
COMPACT	Определяет компактную модель памяти (в программе используются сегмент кода и несколько сегментов данных)
LARGE	Определяет большую модель памяти (в программе используются несколько сегментов кода и несколько сегментов данных)
HUGE	Определяет максимальную модель памяти (в программе используются несколько сегментов кода и несколько сегментов данных)