Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное
учреждение высшего профессионального образования
«Севастопольский государственный университет»

ИССЛЕДОВАНИЕ МЕТОДОВ АДРЕСАЦИИ И ПРОГРАММИРОВАНИЯ АРИФМЕТИЧЕСКИХ И ЛОГИЧЕСКИХ ОПЕРАЦИЙ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к лабораторным работам по дисциплине

Технические средства информационных систем

для студентов, обучающихся по направлению

09.03.02 Информационные системы и технологии

09.03.03 Прикладная информатика

очной и заочной форм обучения

Севастополь
2022

УДК 004.732

Исследование методов адресации и программирования арифметических и логических операций. Методические указания к лабораторным занятиям по дисциплине "Технические средства информационных систем" / Сост. Чернега В.С., Дрозин А.Ю. — Севастополь: Изд-во СевГУ, 2022— 13 с.

Методические указания предназначены для проведения лабораторных работ по дисциплине “ Технические средства информационных систем “. Целью методических указаний является помощь студентам в выполнении лабораторных работ по исследованию архитектуры 16-разрядных процессоров и персональных ЭВМ, а также по программированию различных задач на языке ассемблера процессора Intel 8086. Излагаются краткие теоретические и практические сведения по системе арифметических и логических команд, необходимые для выполнения лабораторных работ, примеры составления программ, требования к содержанию отчета.

Методические указания рассмотрены и утверждены на методическом семинаре и заседании кафедры информационных систем
(протокол № 1 от 30 августа 2022 г.)

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

Рецензент: Кротов К.В., канд. техн. наук, доцент кафедры ИС

Лабораторная работа

ИССЛЕДОВАНИЕ МЕТОДОВ АДРЕСАЦИИ И ПРОГРАММИРОВАНИЯ АРИФМЕТИЧЕСКИХ И ЛОГИЧЕСКИХ ОПЕРАЦИЙ

1. ЦЕЛЬ РАБОТЫ

Изучить основные директивы языка ассемблера, исследовать их воздействие на процесс ассемблирования и формирования листинга программы.

Исследовать особенности функционирования блоков 16-разрядного микропроцессора при выполнении арифметических и логических операций и при использовании различных способов адресации. Приобрести практические навыки программирования на языке ассемблера МП 8086 арифметических и логических операций с применением различных способов адресации.

2. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

2.1 Директивы языка Ассемблер x8086

Допустимые символы языка ассемблера состоят из прописных и строчных букв (латинских), цифр, специальных знаков +, -, *, /, =, (), [], ’, ’’, ., ;, @, &, ?, <, >, % и символов: перевод строки ПС (OAH), возврат каретки ВК (ODH), табуляции (O9H). Любой другой символ воспринимается как пробел. Наименьшей конструкцией модуля является идентификатор – последовательность букв и цифр (не более 31), начинающийся с буквы. К зарезервированным именам относятся имена регистров, операций, псевдокоманд. Модуль представляет собой последовательность операторов языка, записанных в одной строке и заканчивающихся возвратом каретки и переводом строки. Если в первой позиции оператора стоит символ &, то оператор является продолжением предыдущего.

Операторы разделяются на **командные и директивы**. Команды порождают одну машинную команду. Директивы (псевдокоманды) содержат управляющую информацию для ассемблера. Оператор команды имеет вид:

{метка:} {префикс} {мнемоника} {операнд(ы)}; { комментарий}.

Значения *метки* являются текущим значением счетчика в данном сегменте кода, то есть, представляет собой адрес команды. *Префикс* позволяет формировать байты блокировки LOOK или повторения REP. *Мнемоника* идентифицирует тип генерируемой команды. В зависимости от функции команды может быть один операнд, два или ни одного. Более двух операндов указывается в макрокоманде. *Комментарии* поясняют смысл команды.

Директивы ассемблера имеют несколько другой формат:

{имя} директива {операнд(ы)} {; комментарий}

Имя директивы имеет другой смысл по сравнению с меткой и не заканчивается двоеточием. В ряде директив имя отсутствует.

Директивы используются для распределения памяти, связей между модулями, манипуляции с символами и т.д. В отдельных директивах допускаются списки операндов. Операнды могут быть ключевыми словами в директивах PROG, SEGMENT. Для определения макрокоманд используется оператор вида:

MACROCODE имя {операнд(ы)}; {комментарии}

Переменная – это единица данных, имеющая имя. Она имеет три атрибута: **сегмент, смещение** и **тип**. *Сегмент* SEG определяет сегмент, содержащий переменную. *Смещение*_OFFSET, расстояние от начала сегмента до переменной, *тип* – число байтов переменной (1,2 или 4).

Метка, представляющая имя ячейки памяти, имеет атрибутами сегмент, смещение, расстояние. *Константа* отличается от переменной и метки тем, что она определяет только число. Символьные цепочки заключаются в апострофы и обычно имеют длину до 255 знаков.

Для определения и инициализации данных предназначены директивы:

DB - определить байт	(Define Byte)
DW - определить слово	(Define Word)
DD - определить двойное слово	(Define Double Word).

Формат директивы:

имя DX <начальное значение>, [< начальное значение>]

Пример определения переменных:

```
Z1 DB 0ABH            ; один байт, равный AB
Z2 DW 1000H           ;  одно слово, равное 1000
Z3 DD 1235H; 1000H    ; младшее слово 1000, старшее 1235.
```

Для указания произвольного значения ячеек памяти используется символ ? (значение переменной не оговаривается, резервирование ячейки). Для распределения и инициализации нескольких ячеек памяти используется конструкция DUP:

```
DB    100 DUP(0)              ; сто нулевых байтов (Dublicate)
DW     20 DUP(?)              ; 20 слов без значения

ADR   DD    10 DUP(ADR)    ; десять полных адресов
DB    10 DUP(80DUP)      ; десять слов, содержащих 80 пробелов
```

Длина операнда может определяться:

а) кодом команды – в том случае, если используемая команда обрабатывает данные определенной длины, что специально оговаривается;
б) объемом регистров, используемых для хранения операндов (1, 2,4 байта);
в) специальными указателями

byte ptr (1 байт), **word ptr** (2 байта) и **dword ptr** (4 байта),

которые используются в тех случаях, если ***ни один операнд не находится в регистре и размер операнда отличен от размера, определенного директивой объявления данных***. Например:

```
mov byte ptr x, 255
; нас интересует только первый байт слова, хотя x определен как слово.
. . .
x  DW  25
```

В директивах DW, DD допускаются символьные цепочки длиной 1 и 2 символа, например:

```
DW  'K1';   DD  'G'.
```

В выражениях, где запрашивается тип используемой информации, применяются операторы TYPE, LENGTH, SIZE. TYPE (тип) сообщает число байт, отведенных для переменной (1,2,4). LENGTH (длина) определяет число единиц памяти переменной (байт, слов, двойных слов). SIZE (размер) сообщает размер переменной в байтах, причем размер переменной равен длине, умноженной на тип.

В ассемблере вводится понятие логического сегмента, под которым понимается часть программы, которая может включать сегменты для машинного кода, данных и стека. Каждый логический сегмент должен начинаться с директивы **SEGMENT** и заканчиваться директивой **ENDS**. Логическому сегменту присваивается имя, данное программистом, и список параметров (атрибутов), которые не обязательны, но необходимы в случае программы, включающие несколько модулей.

Пример определения простого сегмента:

```
DATA  SEGMENT  [<список атрибутов>]

    F1    DB ?
    F2    DW ?
    F3    DD ?
DATA ENDS
```

До использования сегментного регистра в формировании адреса он должен быть инициализирован, для чего используется имя логического сегмента.

Пример:

```
MOV AX, DATA
MOV DS, AX
```

До использования стека необходимо инициализировать регистры SS и SP. В общем случае в программе первыми командами являются команды инициализации регистров DS, SS, SP. В языке ассемблера 8086 допускаются вложенные сегменты. Это означает, что если определен некоторый сегмент S1 (парой директив SEGMENT и ENDS), затем определяется сегмент S2, а потом в новой директиве SEGMENT снова появляется сегмент с именем S1, то содержащиеся в новом сегменте данные и (или) команды будут автоматически размещаться за операторами, которые находятся в старом сегменте с именем S2. Вложенный сегмент должен заканчиваться раньше внешнего сегмента.

Параметры директивы SEGMENT.

1. **Выравнивание**. Определяет границу начала сегмента. Обычное значение - PARA, по которому сегмент устанавливается на границу параграфа. В этом случае адрес кратен 16. При отсутствии этого операнда ассемблер по умолчанию принимает PARA; адрес сегмента XXX0. Бывает: PAGE=XX00; WORD=XXXE (четная граница); BYTE=XXXX – любая шестнадцатеричная цифра.

2. **Объединение**. Определяет, объединяется ли данный сегмент с другими сегментами в процессе компоновки после ассемблирования. Возможны следующие типы объединений:

STACK, COMMON (общий), PUBLIC (общедоступный), AT- и MEMORY.

Все PUBLIC **- сегменты, имеющие одинаковое имя или класс, загружаются компоновщиком в *смежные области*. Все такие сегменты имеют один общий базовый адрес.**

Для сегментов **COMMON** с одинаковыми именами и классами компоновщик устанавливает один *общий базовый адрес*. При выполнении происходит наложение одного сегмента на другой. Размер общей области определяется самым длинным сегментом.

AT-параграф: он должен быть определен предварительно. Данный операнд обеспечивает определение меток и переменных по фиксированным адресам в фиксированных областях памяти, таких как ROM или таблица векторов прерываний. Например, для определения адреса дисплейного видеобуфера используется

VIDEO_RAM SEGMENT AT 0B800h.

Для сегмента стека используется объединение STACK.

Если программа не должна объединяться с другими программами, то указание типов объединения должно быть опущено (за исключением STACK).

3. Класс. Данный элемент, заключенный в апострофы, используется для группирования относительных сегментов при компоновке. Он может содержать любое имя и используется ассемблером для обработки сегментов, имеющих одинаковые имена и классы.

Типичными примерами являются классы ‘STACK’ и ‘CODE’. (См. П. Абель с. 55 и 396).

Директива ASSUME (присвоить). Необходимая ассемблеру информация о значении сегментных регистров сообщается в директиве ASSUME, которая имеет формат:

ASSUME <SR: базовое значение>, [<SR: базовое значение>]…

Директива ASSUME DS: data1 требуется для транслятора, указывая ему, что сегмент сопоставляется с регистром DS. Описание в директиве ASSUME соответствия сегментного регистра DS сегменту данных избавляет нас от необходимости указывать в каждой строке, содержащей ссылку на имя данных, в каком сегментных регистров находится сегментный адрес этих данных. По умолчанию будет подразумеваться регистр DS. Если регистр ES в директиве ASSUME не описан, его следует в явной форме указывать во всех строках программы, где выполняется адресация к дополнительному сегменту данных:

Mov BX, ES: mas[DI]

Однако, и в этом случае занесение в регистр ES требуемого сегментного адреса должно быть сделано в явном виде:

```
mov ax, data1
mov ex, ax
```

Поле SR содержит имя одного из сегментных регистров CS, DS, SS, ES, а базовое значение указывает на начало области памяти, адресуемой через этот регистр. Одним из наиболее часто используемых типов базового значения является имя сегмента, например

ASSUME DS: DATA

Такая директива сообщает ассемблеру, что к определенным в сегменте DATA переменным можно обратиться через сегментный регистр DS. Директива ASSUME необходима перед использованием сегментных регистров и перед каждой точкой в программе, в которой может модифицироваться содержимое сегментных регистров. Если какой-либо сегмент не будет использован в модуле, то директива будет иметь вид ASSUME NOTHING.

Директива ORG (origin - начало). Основной внутренней переменной ассемблера является счетчик адресов, который при ассемблировании выполняет функцию, аналогичную функции программного счетчика при выполнении программ. Этот счетчик сообщает ассемблеру адрес следующей ячейки памяти (имеется в виду *смещение* в сегменте), которая предназначена для размещения следующего байта команды или данных.

Первое появление директивы <имя> SEGMENT определяет начало сегмента с заданным именем. При этом организуется *новый* счетчик ячеек, в который загружается нулевое значение, так как первый байт в сегменте имеет нулевое смещение. При распределении каждого следующего байта производится инкремент счетчика ячеек.

Директива ENDS с тем же именем отключает данный счетчик до тех пор, пока этот же сегмент не будет открыт еще одной директивой SEGMENT. В этом случае счетчик продолжает счет распределяемых байт со старого значения.

Текущее значение счетчика адресов может быть принудительно изменено программистом с помощью директивы ORG <выражение>. При выполнении директивы ORG вычисляется значение выражение и полученный результат загружается в счетчик ячеек (адресов).

Ассемблирование следующих команд или элемента данных производится по полученному адресу (смещению в текущем сегменте), который изменяется в диапазоне 0-65535 (вычисление осуществляется по модулю 64К).

Директива EQU. Директива позволяет определить символические имена для часто используемых выражений. Формат директивы:

<имя> EQU <выражение>

Поля выражения может определять константы, адреса, регистры и даже мнемокоды макрокоманд. Именам целесообразно придавать содержательный смысл. Допустим, в программе необходимо многократно применять размер таблицы. Тогда его можно определить:

SIZE TABL EQU 100; 100 – размер таблицы.

Затем можно использовать имя SIZE TABL в любой команде, где требуется использовать этот размер. Такой прием улучшает понимание программы, а при расширении таблицы достаточно заменить в директиве EQU число 100 другим размером и произвести повторное ассемблирование программы. Ассемблер автоматически заменит каждое появление в новой команде SIZE TABL новым размером.

Имена, присвоенные директивами EQU, можно использовать в любых операторах исходного модуля, например:

```
MOV CL, LF   ; загрузить в CL значение OA
INC COUNT    ; инкремент содержания регистра CX,
             ; используемого в качестве счетчика.
```

Процедура (замкнутая подпрограмма) представляет собой законченную командную последовательность, которая приводится в действие командой вызова CALL. Процедура является одним из средств разработки модульных программ. Каждая процедура допускает автономную отладку, что позволяет ускорить разработку и отладку всей прикладной программы.

Команда CALL содержит метку одной из команд процедуры. Метка в процедуре, которой передает управление команда CALL, называется *точкой входа процедуры*. Процедура выполняется до тех пор, пока не встретится команда возврата RET.

Процедура обычно разрабатывается как функциональный блок, который формирует набор выходных данных путем преобразования четко определенных входных данных, называемых *параметрами*. Поэтому суть действий с процедурами сводится к передаче им параметров и получении из них результатов. Для организации процедур в языке ассемблера предназначены директивы PROC и

ENDP. Директива PROC отмечает точку входа процедуры, а директива ENDP – окончание процедуры.

Формат этих директив имеет вид:

```
[имя]  PROC  [тип]
. тело процедуры
[имя]  ENDP
```

Имя представляет точку входа в процедуру. *Тип* процедуры (NEAR или FAR) (по умолчанию NEAR) используется ассемблером для определения вида генерируемой команды CALL для вызова процедуры. Если указан тип NEAR, то процедура находится в том же сегменте кода, в котором находятся все команды CALL, вызывающие эту процедуру. В этом случае ассемблер генерирует команду внутрисегментного вызова, то есть машинная команда CALL содержит только смещение точки входа.

Если указан тип FAR, процедура находится в сегменте, отличающимся от того сегмента кода, в котором находятся команды вызова CALL. То есть ассемблер должен генерировать длинную команду межсегментного вызова, которая содержит базу и смещение точки входа. При передаче управления в стеке приходится запоминать, а затем восстанавливать содержимое регистров CS и PC.

В соответствии с двумя форматами команд вызова необходимы и две разновидности команды возврата RET. Ассемблер определяет генерируемую разновидность команды RET на основе типа, указанного в директиве PROC.

2.3 Методы адресации микропроцессорных систем

В процессе программирования МП 8086 используется 7 режимов адресации.

1. **Регистровая адресация**. Операнд находится в одном из регистров общего назначения, а в ряде случаев — в сегментном регистре:

```
mov  dx,bx
add   bx,di
```

2. **Непосредственная адресация**. 8- или 16-битовая константа содержится в команде в качестве источника:

```
mov  al,45
mov  ax,1024
```

3. **Прямая адресация**. Эффективный адрес берется из поля смещения команды. Применяется в основном, если операндом служит метка:

```
table  db 10, 20, 30
.............
mov  ax, table
```

Микропроцессор добавляет адрес к сдвинутому на 4 разряда содержимому регистра DS и получает 20-разрядный адрес операнда.

4. **Косвенная регистровая адресация**. Эффективный адрес содержится в базовом регистре BX, регистре указателя базы BP или индексном регистре (DI или SI). Косвенные регистровые операнды заключают в квадратные скобки:

```
mov ax,[bx]
```

По этой команде в регистр AX загружается содержимое ячейки памяти, адресуемой значением регистра BX.

5. **Базовая адресация**. Эффективный адрес вычисляется с помощью сложения значения сдвига с содержимым одного из базовых регистров BX или BP. Например, при доступе к элементам таблицы адрес начала таблицы предварительно записывается в регистр BX:

```
mov ax,[bx]+8
```

6. **Прямая адресация с индексированием**. Эффективный адрес вычисляется как сумма значений сдвига и одного из индексных регистров (DI или SI). Такой способ адресации целесообразно применять при доступе к элементам таблицы, когда сдвиг указывает на начало таблицы, а индекс - на элемент таблицы.

```
table db 10, 20, 30, 40
.............
mov di,2 ; загрузить в индексный регистр
            ;номер выбираемого байта минус 1
mov al,table[di]  ; загрузить 3 байт таблицы в al
```

7. **Базово-индексная адресация**. Эффективный адрес вычисляется как сумма значений базового регистра, индексного регистра и, возможно, сдвига. Это удобно при работе с двухмерными таблицами: базовый регистр содержит начальный адрес массива, значения сдвига и индексного регистра представляют собой смещения по строке и столбцу.

```
table dw  1024, 1048, 2048,3600
      dw  4100, 5000, 600, 2000
      dw  80, 300, 4000, 5000
   .............
value db 2              ; номер элемента в строке-1
   .............
mov bx,table            ;адрес таблицы
mov di,16               ;адрес начала третьей строки
mov ax,value[bx][di] ;загрузка элемента
table(3,3)=4000
```

Следует иметь в виду, что если в программе есть шестнадцатеричные числа, то чтобы транслятор мог отличить число, начинающееся с буквы от имени переменной, всякое шестнадцатеричное число, начинающееся с буквы, следует предварять нулем.

2.4 Арифметические и логические команды

Команда сложения ADD позволяет выполнять сложение 8- или 16-битовых двоичных чисел в режиме регистр-регистр, регистр-память и память--регистр, причем адресация памяти осуществляется в любом допустимом режиме. Общее представление команды имеет вид

ADD dst, src

т.е. первый операнд складывается со вторым и результат операции замещает первый операнд. Формат команды имеет следующий вид:

ADD mem/reg1,mem/reg2
ADD mem/reg,data

Команда ADC выполняет сложение с переносом. В отличие от команды ADD в операции сложения участвует флаг CF, значение которого прибавляется к младшему биту результата сложения операндов. Формат команды аналогичен формату команды обычного сложения.

Команда инкрементации данных INC увеличивает на 1 содержимое любого общего регистра или ячейки памяти. При переполнении числа CF флаг принимает значение 1. Формат команды:

INC mem/reg

Команда вычитания SUB позволяет производить вычитание 8- или 16-битных двоичных чисел. Общее представление команды имеет вид

SUB dst, src

т.е. второй операнд вычитается из первого и результат операции замещает первый операнд. Формат команды аналогичен формату командам сложения.

Команда SBB выполняет вычитание с заемом. В отличие от команды SUB в операции вычитания участвует флаг CF, значение которого вычитается из младшего бита результата вычитания операндов.

Команда декрементации DEC уменьшает на 1 содержимое любого общего регистра или ячейки памяти. Формат команды:

DEC mem/reg

Команда сравнения CMP выполняет вычитание второго операнда из первого, но нигде не запоминает результат операции и влияет только на состояние флажков. Формат команды:

CMP mem/reg1,mem/reg2
CMP mem/reg,data

Операция умножения для беззнаковых данных выполняется командой **MUL**, а для знаковых - IMUL (Integer MULtiplication - умножение целых чисел).

"Б а й т н а б а й т". Множимое находится в регистре AL, а множитель в байте памяти или в однобайтовом регистре. После умножения произведение находится в регистре AX. Операция игнорирует и стирает любые данные, которые находились в регистре AH.

"С л о в о н а с л о в о". Множимое помещается в регистр AX, а множитель - в ячейках памяти или в регистре. После умножения произведение находится в двойном слове, для которого используются два регистра: старшая (левая) часть произведения заносится в регистр DX, а младшая (правая) часть в регистр AX. Операция игнорирует и стирает любые данные, которые находились в регистре DX.

Формат команды:

MUL reg
MUL mem

Если поле операнда определено как байт (DB), то операция предполагает умножение содержимого AL на значение байта из поляоперанда. Если поле операнда определено как слово (DW), то операция предполагает умножение содержимого AX на значение слова из поляоперанда. Если множитель находится в регистре, то длина регистра определяет тип операции, как это показано ниже:

```
MUL  CL  ; Байт-множитель: множимое в AL, произведение в AX
MUL  BX  ; Слово-множитель: множимое в AX, произведение в DX:AX
```

Команда IMUL аналогична команде MUL, но сомножители и произведение интерпретируются как знаковые двоичные числа в дополнительном коде. Формат команды:

IMUL reg
IMUL mem

Команды деления.

Микропроцессор 8086 имеет две команды деления: для беззнаковых и для знаковых двоичных чисел. Деление десятичных чисел также требует использования специальных команд коррекции.

Команда деления беззнаковых чисел DIV производит деление содержимого аккумулятора и его расширения на содержимое адресуемого операнда.

При делении 16-битного делимого на 8-битный делитель делимое помещают в регистр AX. В результате выполнения операции частное формируется в регистре AL, а остаток - в AH.

При делении 32-битного делимого на 16-битный делитель старшая часть делимого помещается в регистр DX, а младшая - в AX. В результате выполнения операции частное формируется в регистре AX, а остаток - в DX.

При делении на 0 автоматически происходит прерывание и переход к специальной программе обработки. Формат команды:

DIV reg
DIV mem

Команда IDIV аналогична команде DIV, но делимое, делитель и частное интерпретируются как знаковые двоичные числа в дополнительном коде. Формат команды:

IDIV reg
IDIV mem

Команды логических операций.

Логические операции представлены командами NOT (инверсия), AND (конъюнкция), OR (дизъюнкция), XOR (исключающее ИЛИ) и командой TEST. Последняя выполняет конъюнкцию операндов, но не изменяет их значений. Все логические операции являются поразрядными, т.е. выполняются независимо для всех бит операндов.

Бинарные команды AND, OR, XOR и TEST воздействуют на флажки OF, SF, ZF, PF и CF. Унарная операция NOT не влияет на состояние флажков.

Форматы команд:

AND mem/reg1,mem/reg2
AND mem/reg,data
OR mem/reg1,mem/reg2
OR mem/reg,data
XOR mem/reg1,mem/reg2
XOR mem/reg,data
TEST mem/reg1,mem/reg2
TEST mem/reg,data
NOT mem/reg

Более подробное описание команд и результат воздействия их на содержимое регистров общего назначения и ячеек памяти находится в закладке help эмулятора лабораторной установки emu8086.

3. ОПИСАНИЕ ЛАБОРАТОРНОЙ УСТАНОВКИ

Лабораторный стенд для исследования архитектуры и способов программирования на языке ассемблера 16-разрядных микропроцессоров состоит из персонального компьютера, на котором установлен программный эмулятор 16-

разрядного микропроцессора типа Intel 8086 (отечественный аналог МП КР1810) emu8086. Эмулятор отображает на экране персонального компьютера программную модель исследуемого процессора, а также позволяет создавать и редактировать тексты программ на языке ассемблера МП 8086, выполнять их ассемблирование и исследование процессов модификации регистров процессора, дампов памяти и портов в пошаговом и реальном режимах отладки программ. Работа с эмулятором подробно описана в методических указаниях по предыдущей работе.

4. ПРОГРАММА ИССЛЕДОВАНИЙ

4.1. Изучить основные директивы ассемблера и их воздействие на процесс ассемблирования и формирования листинга программы. Повторить команды пересылки данных, а также команды арифметических и логических операций (выполняется в процессе домашней подготовки к лабораторной работе).

4.2. Изучить методы адресации, используемые в 16-разрядных процессорах и особенности оформления программ в exe- и com-форматах (выполняется во время домашней подготовке к работе).

4.3. Составит программу в com и exe форматах, осуществляющей вычисление выражения, приведенного в приложении, согласно вашему варианту. Номер варианта определяется последней цифрой номера зачетной книжки.

4. Произвести отладку разработанных программ в пошаговом режиме и проследить за изменениями содержимого регистров

5. Рассчитать время выполнения программ.

5. СОДЕРЖАНИЕ ОТЧЕТА

4.1 Цель и программа работы.

4.2 Текст и листинг ассемблерной программы с комментариями для заданного варианта.

4.3 Выводы по работе.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Каково различие между директивой и командой?

6.2. Назовите директивы определения данных ассемблера и поясните механизм их действия.

6.3. Какие директивы применяются для оформления процедур?

6.4. Какие типы сегментов используются в ассемблерных программах и каково их назначение?

6.5. Поясните назначение параметров выравнивания и объединения, используемых в директивах SEGMENT.

6.6. Когда и в каких случаях применяется директива ORG?

6.7. Что конкретно подразумевает директива END, если она завершает: а) программу, б) процедуру, в) сегмент?

6.8. Какие операции необходимо произвести в процессоре до начала выполнения программы?

6.9. Назовите команды арифметических операций и поясните использование регистров процессора при каждой операции.

6.10.С какой целью в начале кодового сегмента в стек заносится содержимое сегментного регистра DS, а затем нулевое значение?

6.11.Каково назначение директивы ASSUME?

6.12.Расскажите об особенностях размещения в памяти ЭВМ программ с расширениями .exe и .com.

6.13.Нарисуйте схему подключения 16-разрядного порта к МП 1810ВМ86, если в наличии имеются только микросхемы 580ВВ55 или 580ВА86.

6.14.Каково назначение вывода M/IO в МП 8086 и нарисуйте схему подключения устройств с его использованием.

6.15.Какая информация хранится в заголовке .exe-программы, его назначение и размер?

6.16.Зачем к исполняемому модулю добавляется префикс программного сегмента, какой его размер и какая информация в нем хранится?

6.17.Расскажите о методах адресации, используемых в МП-системах, и объясните в каких случаях целесообразно использование этих методов?

6.18.Объясните особенности использования строковых команд.

6.19.Каким образом можно изменять направление просмотра строк?

6.20.Чем отличаются команды CMPS и SCAS?

6.21.Как обеспечить ввод данных с группы 16-разрядных портов с максимальной скоростью?

6.22.В чем состоит суть защищенного режима работы процессора и как осуществляется защита?

6.23.Что такое дескриптор сегмента, из каких частей он состоит и как используется при защите памяти?

6.24.Как организуется виртуальная память и как используется дескриптор для ее поддержки?

6.25.Расскажите об архитектуре 16-разрядного процессора второго поколения и приведите его схему.

6.26.Расскажите о регистрах 16-разрядного процессора второго поколения и особенностях их использования в защищенном режиме.

6.27.Расскажите о многозадачном режиме работы процессора, составе и назначении сегмента состояния задачи.

7. СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

7.1. Абель П. Язык Ассемблера для IBM PC и программирования / Пер. с англ. Ю.В.Сальникова. — М.: Высш. школа, 1992. — 447 с.Новиков

Ю.В. Основы микропроцессорной техники: Учебное пособие/Ю.В. Новиков, П.К. Скоробогатов. — М.: Интернет-университет информационных технологий; БИНОМ, 2006. — 359 с.

7.2. Макуха В.К.. Микропроцессорные системы и персональные компьютеры: учебник для вузов/В.к. Макуха, В.А. Микерин. – М.: Изд-во Юрайт, 2022. – 156 с. https://www.urait.ru/book/mikroprocessornye-sistemy-i-personalnye-kompyutery-492153

7.3. Программирование на ассемблере для процессоров персонального компьютера / М.К. Маркелов, Л.С. Гурьянова, А.С. Ишков, А.С. Колдов, С.В. Волков.— Пенза: ПГУ, 2013 .— ISBN 978 -5-94170-537-5 http://rucont.ru/efd/210624?cldren=0

7.4. Новожилов О.П. Архитектура ЭВМ и систем в 2 Ч. Часть 1. Учебное пособие для академического бакалавриата / О.П. Новожилов. – М.: Изд-во Юрайт, 2019. – 276 с. https://urait.ru/viewer/arhitektura-evm-i-sistem-v-2-ch-chast-1-494314#page/1

7.5. Чернега В.С. Архитектура информационных систем. Конспект лекций / В.С. Чернега. – Севастополь: Изд-во СевГУ, 2019 – 160 с.

Приложение

Варианты заданий

Вариант	Выражение
1	X= (2A + B) / 4 - C / 2 + 168
2	X= 6 (A - 2B + C / 4) + 10
3	X= 5 (A - B) + C * 4
4	X= (C + 2A) * B/4 + 38)
5	X= A/3 - 3 (A + B) + C * 4
6	X= 3(A - 2B) +50 – C / 2
7	X= (3A + 2B) - C / 4 + 217
8	X= 3(C - 2A) + (B - C + 1) / 2
9	X= (2A + B) / 4 - C / 2 + 168
10	X= 6 (A - 2B + C / 4) + 10

Заказ № _____ от «___» _____ 2022 г. Тираж _____ экз.

Изд-во СевГУ