ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

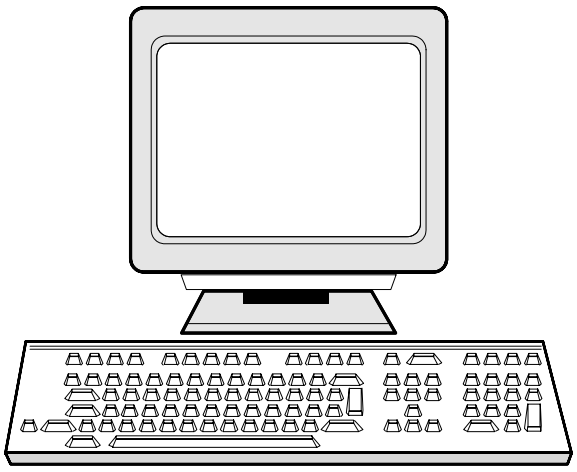
Государственное образовательное учреждение высшего профессионального образования

УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Архитектура ЭВМ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

ДЛЯ СТУДЕНТОВ СПЕЦИАЛЬНОСТИ
ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ



Ульяновск

2008

УДК 004.2 (076)
ББК 32.973.26-04
А

Рецензент – профессор кафедры «Вычислительная техника» факультета информационных систем и технологий Ульяновского государственного технического университета доктор технических наук В. Н. Негода.

Одобрено секцией методических пособий научно-методического совета университета

А **Архитектура ЭВМ.** Задания и примеры выполнения лабораторных работ: Методические указания для студентов специальности 23020165 / Сост.: А. Е. Докторов, Е. А. Докторова – Ульяновск: УлГТУ, 2008. – 32 с.

Методические указания составлены в соответствии с учебным планом специальности 23020165 «Информационные системы и технологии». Преследуют цель ориентировать студентов на содержание и порядок выполнения лабораторных работ по архитектуре ЭВМ. Даются начальные сведения по темам, примеры оформления и выполнения лабораторных заданий.

Методические указания подготовлены на кафедре ИВК.

УДК 004.02 (076)
ББК 32.973.26-04

ВВЕДЕНИЕ

Под *архитектурой ЭВМ* (или *архитектурой системы команд*), чаще всего принято определять те средства процессора, которые видны и доступны программисту.

Системой команд процессора называют полный перечень команд, которые способен выполнять данный процессор.

Данные, доступные программисту, могут храниться в *памяти* (внешней по отношению к процессору) и *регистрах* (внутренней памяти процессора).

При обращении к *оперативной памяти* в языках высокого уровня программист оперирует понятием *переменной*, а процессор – *адресом* размещения этой переменной в оперативной памяти.

При обращении к *регистру* процессора программист определяет его по обозначению (имени), заданном разработчиком процессора, а процессор – по номеру (или адресу).

Разработчики процессоров могут не разделять регистры по их назначению, имена таких регистров чаще всего отличаются номерами, например: R1, R2, R3 и т.д. Такие регистры называют регистрами общего назначения.

Исторически сложилось, что регистры процессоров фирмы Intel имеют различия по назначению и, соответственно, имеют разные обозначения. Когда регистры имели 8 двоичных разрядов (процессор был восьмиразрядный), регистры обозначались одной буквой. Например, у процессора Intel 8080 были следующие обозначения регистров:

A – аккумулятор (accumulator), предназначенный для хранения данных при выполнении арифметических и логических операций;

B – базовый регистр (base), используемый для задания смещения адреса по базе;

C – счетчик (count), для организации циклов;

D – регистр для хранения данных (data).

В соответствии с разрядностью шины данных (16 разрядов) у процессора Intel 8086 есть свой набор регистров. Чтобы показать, что разрядность регистров стала больше, по сравнению с восьмиразрядными процессорами, к имени регистра была добавлена буква «X» (eXtended – расширенный). Получаются, соответственно, имена: AX, BX, CX, DX. У процессора Intel 8086 можно выбрать не весь регистр, а только младшую (Low) или старшую (High) часть регистра. Соответствующими будут и имена: AH, AL, BH, BL.

У старшего поколения процессоров, с 32-х разрядной шиной, кроме всего сказанного для процессора Intel 8086, наименование 32-х разрядных регистров стало ещё длиннее: EAX, EBX, ECX и т.д.

Лабораторными заданиями предусмотрено изучение архитектуры процессора персонального компьютера, точнее говоря, в большей степени будет изучаться система команд процессора персонального компьютера во встроенном ассемблере Free Pascal, при этом за основу взяты команды для процессора Intel 8086.

1. АРХИТЕКТУРА ПРОЦЕССОРА

Наименьшей единицей данных, с которой работает процессор, является бит (bit). Значением бита может быть либо ноль, либо единица. Группа из восьми битов называется байтом (Byte) и представляет собой наименьшую адресуемую единицу – ячейку. Биты в байте нумеруют справа налево цифрами 0...7. Двухбайтовое поле образует шестнадцатиразрядное машинное слово (Word), биты в котором нумеруются от 0 до 15 справа налево. Байт с меньшим адресом считается младшим. Аналогично представляются 32-х разрядные слова.

В процессоре принята двоичная система представления данных. Числовые данные кодируются в соответствии с двоичной арифметикой. Отрицательные числа представляются в дополнительном коде. Для удобства представления данных используется шестнадцатеричная система счисления. Принято двоичные числа сопровождать латинской буквой B или b, например, 101B, а шестнадцатеричные – буквой H или h на конце. Если число начинается с буквы, то обязательной является постановка нуля впереди, например, 0BA8H.

Регистры

В интегрированной среде Free Pascal можно просмотреть содержимое пятнадцати 32-разрядных регистров процессора, которые используются для управления исполнением команд, адресации и выполнения арифметических операций. Регистр, содержащий одно слово, адресуется по имени.

Регистры сегмента CS, DS, SS, ES, FS и GS.

Регистр CS – содержит начальный адрес сегмента кода.
Регистр DS – содержит начальный адрес сегмента данных.
Регистр SS – содержит начальный адрес регистра стека.
Регистры ES, FS и GS – дополнительные сегментные регистры.

Регистры общего назначения EAX, EBX, ECX и EDX

Регистры общего назначения являются основными рабочими регистрами ассемблерных программ. Их отличает то, что к ним можно адресоваться одним 32-х разрядным словом, 16-и разрядным словом или однобайтовым кодом. Например, у регистра EAX можно использовать все 32 разряда, тогда будет использоваться всё его имя EAX. Два младших байта (16 разрядов), тогда его имя – AX. Из шестнадцатиразрядного регистра AX также можно выделить две части: младший байт AL и старший байт AH. Аналогично могут по частям рассматриваться и другие указанные регистры.

Регистр EAX –аккумулятор, используется во всех операциях ввода/вывода, в некоторых операциях со строками и в некоторых арифметических операциях.

Регистр EBX – базовый регистр, единственный из регистров общего назначения, используемый в косвенной адресации. Кроме того, регистр EBX используется при вычислениях.

Регистр EDX – регистр данных. Используется в некоторых операциях ввода/вывода, в операциях умножения и деления больших чисел совместно с регистром EAX.

Любой из регистров общего назначения может быть использован для суммирования или вычитания.

Регистры указателя ESP и EBP

Регистры указателя используются для обращения к данным в сегменте стека. Могут использоваться все 32 разряда или только младшие 16 разрядов, тогда имя регистра используется без буквы «E».

Регистр ESP – указатель стека (stack pointer). Используется для определения адреса вершины стека.

Регистр EBP – указатель базы (base pointer). Обеспечивает ссылки на параметры (данные и адреса, передаваемые через стек).

Индексные регистры ESI и EDI

Индексные регистры используются для адресации, а также для выполнения операций сложения и вычитания. В них могут быть использованы все 32 разряда или только младшие 16 разрядов, тогда имя регистра используется без буквы «E».

Регистр ESI – индекс источника (source index). Используется в некоторых операциях со строками или символами.

Регистр EDI – индекс приемника (destination index). Используется в тех же операциях, что и регистр ESI.

Регистр указателя команд EIP

Регистр IEP используется для выборки очередной команды программы с целью ее исполнения.

Регистр флагов Flags

Регистр Flags содержит девять активных битов (из 16), которые отражают состояние процессора и результаты выполнения машинных команд.

Биты:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Флаг					O	D	I	T	S	Z		A		P		C

Флаги

O (переполнения) – равен 1, если возникает арифметическое переполнение, например, при сложении числа 01111111B (127 десятичное) с числом 00000001B, получится число 10000000B (-128 десятичное), то есть семь разрядов, используемых для представления абсолютной величины числа, переполнились, и был задействован знаковый разряд.

D (направления) – устанавливается в 1 для автоматического декремента в командах обработки строк, и в 0 для инкремента.

I (разрешения прерывания) – прерывания разрешены, если I=1. Если I=0, то распознаются лишь немаскированные прерывания.

T (трассировки) – если T=1, то процессор переходит состояние прерывания INT 3 после выполнения каждой команды.

SF(знака) – S=1, когда старший бит результата равен 1. Иными словами, S=0 для положительных чисел, и S=1 для отрицательных чисел

Z (нулевого результата) – Z=1, если результат равен нулю.

A (дополнительный флаг переноса) – этот флаг устанавливается в 1 во время выполнения команд десятичного сложения и вычитания при необходимости выполнения переноса или заёма между полубайтами.

P (четности) – этот флаг устанавливается в 1, если результат имеет четное число единиц.

C (переноса) – этот флаг устанавливается в 1, если имеет место перенос или заём из старшего бита результата; он полезен для произведения операций над числами длиной в несколько слов, которые сопряжены с переносами и заемами из слова в слово.

Сегменты

Данный параграф в полной мере справедлив только для процессоров с шиной адреса менее 32 разрядов. У 32-х разрядного процессора нет необходимости в формировании полного 32-х разрядного адреса ячейки памяти с помощью каких либо дополнительных регистров.
Сегментом называется область памяти, которая начинается на границе *параграфа*, то есть в любой точке, адрес которой кратен 16 (восемь младших битов равны нулю). Существуют три основных типа сегментов:

- сегмент кода (CS)– содержит машинные команды;
- сегмент данных (DS) – содержит данные;
- сегмент стека (SS) – содержит адреса возврата в точку вызова подпрограмм, локальные переменные и параметры значения.

Каждый из упомянутых регистров содержит адрес начала сегмента (базовый адрес). Для того чтобы выполнить обращение к данным по любому адресу процессор выполняет суммирование адреса, записанного в регистре сегмента DS, со смещением. При этом содержимое регистра DS сдвигается на четыре двоичных разряда влево, чтобы результирующий адрес занимал 20 позиций (для процессора 8086), что и позволяет адресовать 1 Мбайт памяти (2^{20} = 1048586). Например, если в регистре DS было шестнадцатеричное число 045Fh, после сдвига влево на 4 двоичных разряда оно примет вид 045F0h. К полученному числу прибавляется смещение (например, 0032h), и получается исполнительный (или эффективный) адрес равным 04622h.

Режимы адресации

Режимы адресации приведены в соответствии с возможностями встроенного ассемблера Free Pascal. В колонке «Режим адресации» приведено наименование режима. В колонке «Формат адреса», что используется в качестве операнда. В колонке «Стандартный сегментный регистр» – в каком сегменте по умолчанию располагаются данные.

Режим адресации	Формат адреса	Стандартный сегментный регистр
Регистровая	регистр (указывается имя регистра)	Нет
Непосредственная	данные (указывается число)	Нет
Прямая	переменная (указывается имя переменной)	DS
Косвенная регистровая	[EBX]	DS
	[EBP]	SS
	[EDI]	DS
	[ESI]	DS
Косвенная регистровая со смещением	[EBX + смещение]	DS
	[EBP + смещение]	SS
	[EDI + смещение]	DS
	[ESI + смещение]	DS
Косвенная регистровая по базе со смещением и с индексированием	[BX + DI + смещение]	DS
	[BX + SI + смещение]	SS
	[BP + DI + смещение]	SS
Строковые команды	исходный адрес	DS:SI
	место назначения	ES:DI

Стеки

Стек – это структура данных типа LIFO (Last Input First Output, «последний пришел – первый ушел»). Наиболее важное использование стека связано с процедурами. Стек обычно рассчитан на косвенную адресацию через регистр ESP – указатель стека. При включении элементов в стек производится автоматический декремент указателя стека, а при извлечении – инкремент, то есть стек всегда «растет» в сторону меньших адресов памяти. Адрес последнего включенного в стек элемента называется вершиной стека (TOS – Top of Stack).

Физический адрес стека формируется из ESP и SS или EBP и SS, причем ESP служит неявным указателем стека для всех операций включения и извлечения, а SS – сегментным регистром стека. Содержимое SS называется базой стека. Первоначальное содержимое ESP считается наибольшим смещением, которого может достигать стек. Регистр EBP предназначен, главным образом, для произвольных обращений к стеку.

2. СИСТЕМА КОМАНД ПРОЦЕССОРА

Из всего набора команд процессора в лабораторных заданиях предусмотрено рассмотрение следующих команд:

ПЕРЕСЫЛКА ДАННЫХ

MOV PUSH POP XCHG PUSHF POPF
XLAT LEA LDS LES LAHF SAHF

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

ADD ADC INC SUB SBB DEC
CMP MUL IMUL DIV IDIV NEG
CBW CWD

ЛОГИЧЕСКИЕ ОПЕРАЦИИ

NOT SHL/SAL SHR SAR ROL ROR
RCL RCR AND TEST OR XOR

ОБРАБОТКА БЛОКОВ ДАННЫХ

REP MOVS CMPS SCAS LODS STOS
CMPB CMPSW LODSB LODSW MOVSB MOVSB
MOVSW REPE REPNZ SCASB SCASW STOSB
STOSW

КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ

CALL JMP RET

КОМАНДЫ УСЛОВНОГО ПЕРЕХОДА

JA	JLE	JNL	JS
JAE	JNA	JNLE	JZ
JB	JNAE	JNO	LOOP
JBE	JNB	JNP	LOOPE
JC	JNBE	JNS	LOOPNE
JCXZ	JNC	JNZ	LOOPNZ
JE	JNE	JO	LOOPZ
JG	JNE	JP	JGE
JL	JPE	JNGE	JPO

УПРАВЛЕНИЕ СОСТОЯНИЕМ ПРОЦЕССОРА

CLC CMC STC CLD STD NOP

Подробное описание каждой из команд приводится ниже. Каждая запись этого списка содержит информацию о том, какие флаги из регистра FLAGS процессора изменяются.

Поскольку регистр FLAGS содержит всего 9 флагов, эту информацию можно выдать в компактной форме, например:

Флаги:	O	D	I	T	S	Z	A	P	C
	0				*	*	?	*	0 ,

где приняты следующие обозначения флагов:

? – не определен после операции;

* – изменился в зависимости от результатов выполнения команды;

0 – всегда сброшен;

1 – всегда установлен.

Операнды

В этом поле приводится список возможных операндов и способы адресации для каждой команды. В ассемблере принято каждую команду размещать в отдельной строке. Формат записи команд следующий:

имя_команды операнд_приёмник операнд_источник

Причём в зависимости от команды:

1) операндов может не быть совсем;
2) операнд может быть один;
3) операндов может быть два.

Сколько у каждой команды операндов, указано в описании команд, приведённом ниже. Например, есть такие строки описаний (соответственно без операндов, с одним операндом, с двумя операндами):

Команда: LAHF
Команда: POP destination
Команда: LDS destination, source

Слово «source» – это источник, то есть то место, откуда берётся число, «destination» – приёмник (куда поступают данные).

Последнее замечание, перед тем как перейти к краткому описанию команд. За основу приняты команды процессора Intel 8086. В описании даны некоторые поправки для 32-разрядного процессора.

2.1 Команды пересылки данных

LAHF Загрузка AH из регистра флагов

Флаги не меняются.
Команда: LAHF .
Логика: биты регистра AH 7 6 4 2 0 заполняются значениями битов регистра флагов FLAGS: S Z A P C, соответственно .

LDS Загрузка указателя с использованием DS

Флаги не меняются.
Команда: LDS destination, source.
Логика: DS = (source) destination = (source + 2) .
Команда LDS загружает в два регистра 32-битный указатель, расположенный в памяти по адресу source. При этом в сегментный регистр DS заносится 0 (для Free Pascal), а в базовый регистр destination – указатель. В качестве операнда destination может выступать любой 32-битный регистр, кроме сегментных.

LEA Загрузка исполнительного адреса

Флаги не меняются.
Команда: LEA destination, source.
Логика: destination = Addr (source).

Команда LEA присваивает значение адреса операнда source (а не его значение !) операнду destination. Операнд source должен быть ссылкой на память (переменная), а в качестве операнда destination может выступать любой 32-битный регистр, кроме сегментных.

LES Загрузка указателя с использованием ES

Флаги не меняются
Команда: LES destination, source.
Логика: ES = (source)
destination = (source + 2).
Команда LES загружает в два регистра 32-битный указатель, расположенный в памяти по адресу source. При этом в сегментный регистр ES заносится 0, а в базовый регистр destination – указатель. В качестве операнда destination может выступать любой 32-битный регистр, кроме сегментных.

MOV Пересылка (байта, 16- или 32-разрядного слова)

Флаги не меняются.
Команда: MOV destination, source.
Логика: destination = source .
MOV пересылает по адресу destination байт или слово, находящееся по адресу source.

POP выборка 16- или 32-разрядного слова из стека

Флаги не меняются.
Команда: POP destination.
Логика: destination = (SP)
SP = SP + 2 для 16-разрядного слова,
SP = SP + 4 для 32-разрядного слова.

POPF пересылка слова из стека в регистр FLAGS

Команда: POPF .
Логика: flag-register = (SP)
SP = SP + 4 .

PUSH загрузка 16- или 32-разрядного слова в стек

Флаги не меняются.
Команда: PUSH source.
Логика: SP = SP – 2(4)
(SP) = source .

PUSHF загрузка содержимого регистра флагов в стек

Флаги не меняются.
Команда: PUSHF .
Логика: SP = SP - 4
(SP) = flag-register .

SAHF пересылкал регистра AH в регистр флагов

Флаги не меняются.
Команда: SAHF .

Логика: биты регистра флагов FLAGS : S Z A P C
биты регистра AH: 7 6 4 2 0 .

XCHG обмен значениями

Флаги не меняются.
Команда: XCHG destination, source .
Логика: destination <--> source .
Команда XCHG обменивает значения своих операндов, которые могут быть байтами или словами (16-ти и 32-х разрядными).

XLAT кодирование AL по таблице

Флаги не меняются.
Команда: XLAT
Логика: AL = (BX + AL).

2.2. Арифметические операции

ADC Сложение с переносом

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	*	*	*

Команда: ADC destination, source.
Логика: destination = destination + source + C (содержимое флага C)

ADD Сложение

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	*	*	*

Команда: ADD destination, source.
Логика: destination = destination + source .

CBW Преобразование байта в слово

Флаги не меняются.
Команда: CBW.
CBW расширяет бит знака регистра AL в регистр AH. Эта команда переводит байтовую величину со знаком в эквивалентное ей слово со знаком.

CMP Сравнение

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	*	*	*

Команда: CMP destination, source.
Логика: Установка флагов в соответствии с результатом (destination - source)
CMP сравнивает два числа, вычитая операнд source из операнда destination, и изменяет значения флагов. CMP не изменяет сами операнды. Операндами могут быть байты или слова .

CWD Преобразование слова в двойное слово

Флаги не меняются.
Команда: CWD .
CWD расширяет бит знака регистра AX на весь регистр DX.

Эта команда генерирует двойное слово, эквивалентное числу со знаком, находящемуся в регистре AX.

DEC Декремент

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	*	*	

Команда: DEC destination.
Логика: destination = destination – 1. (флаг C не меняется!)

DIV Деление без учета знака

Флаги:	O	D	I	T	S	Z	A	P	C
	?				?	?	?	?	?

Команда: DIV source.
Логика: AL = AX / source; операнд source – байт
AH = остаток
или
AX = DX:AX / source ; операнд source – слово
DX = остаток.

IDIV Деление с учетом знака

Флаги:	O	D	I	T	S	Z	A	P	C
	?				?	?	?	?	?

Команда: IDIV source .
Логика: AL = AX / source; операнд source – байт
AH = остаток
или
AX = DX:AX / source ;операнд source – слово
DX = остаток.

IMUL Умножение с учетом знака

Флаги:	O	D	I	T	S	Z	A	P	C
	*				?	?	?	?	*

Команда: IMUL source.
Логика: AX = AL * source; операнд source – байт
или
DX:AX = AX * source ; операнд source - слово.

INC Инкремент

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	*	*	

Команда: INC destination.
Логика: destination = destination + 1 . (флаг C не меняется!)

MUL Умножение без учета знака

Флаги:	O	D	I	T	S	Z	A	P	C
	*				?	?	?	?	*

Команда: MUL source.

Логика: AX = AL * source; операнд source - байт
или
DX:AX = AX * source ;операнд source - слово .

NEG Получение дополнительного кода

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	*	*	*

Команда: NEG destination.
Логика: destination = - destination; дополнительный код.

SBB Вычитание с заёмом

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	*	*	*

Команда: SBB destination, source.
Логика: destination = destination - source - CF.

SUB Вычитание

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	*	*	*

Команда: SUB destination, source
Логика: destination = destination - source.
Команда SUB вычитает операнд source из операнда destination и засылает результат по адресу destination.

2.3. Логические операции

AND Логическое умножение

Флаги:	O	D	I	T	S	Z	A	P	C
	0				*	*	?	*	0

Команда: AND destination, source
Логика: destination = destination AND source.

NOT Логическое отрицание

Флаги не меняются .
Команда: NOT destination
Логика: destination = NOT(destination).

OR Логическое сложение

Флаги:	O	D	I	T	S	Z	A	P	C
	0				*	*	?	*	0

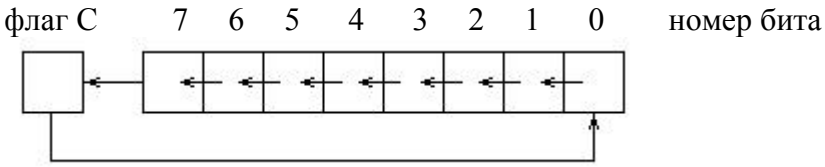
Команда: OR destination, source.
Логика: destination = destination OR source .

RCL Циклический сдвиг влево через флаг C

Флаги:	O	D	I	T	S	Z	A	P	C
	*								*

Команда: RCL destination, count

Логика: Команда RCL сдвигает слово или байт, находящийся по адресу destination, влево на число битовых позиций, определяемое вторым операндом, COUNT. Бит, который выскакивает за левый предел операнда destination, заносится во флаг переноса C, а старое значение C осуществляет ротацию, в том смысле, что оно заносится в освободившийся крайний правый бит операнда destination. Схема сдвига для одного байта:



RCR Циклический сдвиг вправо через флаг C

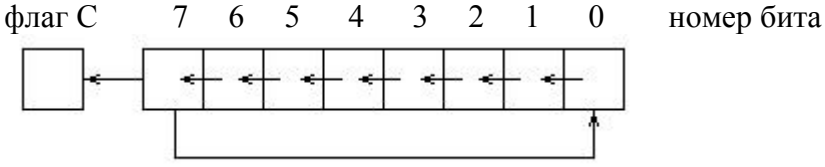
Флаги:	O	D	I	T	S	Z	A	P	C
	*								*

Команда: RCR destination, count
Логика: Аналогична как для команды RCL, но сдвиг идёт вправо, и во флаг C будет заноситься значение из младшего бита, а старое значение из флага C – в старший бит.

ROL Циклический сдвиг влево

Флаги:	O	D	I	T	S	Z	A	P	C
	*								*

Команда: ROL destination, count
Логика: Если COUNT не равен 1, то признак переполнения O не определен. Если же COUNT равен 1, тогда в флаг O заносится результат выполнения операции исключающего или, примененной к 2 старшим битам исходного значения операнда destination. Старое значение старшего бита копируется и в младший бит, и во флаг C. Схема сдвига для одного байта:



ROR Циклический сдвиг вправо

Флаги:	O	D	I	T	S	Z	A	P	C
	*								*

Команда: ROR destination, count
Логика: Аналогична как для команды ROL, но сдвиг идёт вправо. Во флаге C будет старое значение младшего бита, и оно же – в старшем бите.

TEST Проверка битов

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	?	*	*

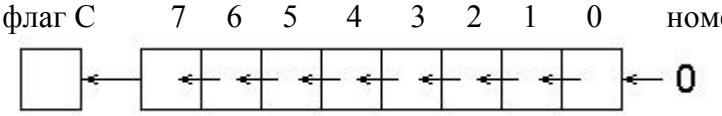
Команда: TEST destination, source

Логика: Действует аналогично команде AND, но не меняет результирующий операнд.

SAL Арифметический сдвиг влево / SHL Логический сдвиг влево

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	?	*	*

Команда: SAL destination, count
Логика: Если COUNT не равен 1, то флаг переполнения O не определен. Если же COUNT равен 1, тогда O=0, если 2 старших бита исходного значения операнда destination совпадали, иначе O=1. Младший бит заполняется нулём.
Схема сдвига для одного байта:



SHR Логический сдвиг вправо

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	?	*	*

Команда: SHR destination, count
Логика: Аналогична команде SHL, но сдвиг идёт вправо, и во флаг C будет скопироваться содержимое младшего бита. Старший бит заполняется нулём.

SAR Арифметический сдвиг вправо

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	?	*	*

Команда: SAR destination, count
Логика: Аналогична команде SHR, но в старшем бите значение не меняется .

XOR Исключающее ИЛИ

Флаги:	O	D	I	T	S	Z	A	P	C
	0				*	*	*	*	0

Команда: XOR destination, source .
Логика: destination = destination XOR source .

2.4. Обработка блоков данных

Для всех команды обработки слов учитывается значение флага направления D. Если D=0, тогда после выполнения команды происходит увеличение регистров SI и DI на 1 для байтовых команд и на 2 для команд, обрабатывающих слово. Если D=1, тогда после выполнения команды происходит уменьшение тех же регистров на 1 для байтовых команд (на 2 при обработке слов).

CMPSB Сравнение строки байтов (CMPSW Сравнение строки слов)

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	*	*	*

Команды: CMPSB и CMPSW
Логика: CMP (DS:SI), (ES:DI) ; только устанавливает флаги

LODSB Загрузка строки из байтов (LODSW Загрузка строки из слов)

Флаги не меняются.
Команда: LODSB или LODSW
Логика: AL = (DS:SI) (или AX = (DS:SI)).

MOVSB Пересылка строки из байтов (MOVSW Пересылка строки из слов)

Флаги не меняются.
Команда: MOVSB или MOVSW
Логика: (ES:DI) = (DS:SI)

REP Повтор

Флаги не меняются.
Команда: REP команда_обработки_строк
Логика: Пока CX <> 0 выполнить команду обработки строк, и CX:=CX-1

REPE Повторять пока равно

Флаги не меняются.
Команда: REPE команда_обработки_строк
Логика: Пока (CX <> 0) и (флаг Z=1) выполнить команду обработки строк, и CX:=CX-1

REPNE Повторять пока не равно или REPNZ

Флаги не меняются.
Команда: REPNE команда_обработки_строк
Логика: Пока (CX <> 0) и (флаг Z=0) выполнить команду обработки строк, и CX:=CX-1

SCASB Просмотр строки из байтов (SCASW просмотр строки из слов)

Флаги:	O	D	I	T	S	Z	A	P	C
	*				*	*	*	*	*

Команда: SCASB или SCASW
Логика: Сравнивает содержимое AL и (ES:DI); или AX и (ES:DI);
только устанавливает флаги.

STOSB Запись в строку из байтов (STOSW Запись в строку из слов)

Флаги не меняются.
Команда: STOSB
Логика: (ES:DI) = AL или (ES:DI) = AX

2.5 Команды передачи управления

CALL Вызов подпрограммы

Флаги не меняются.
Команда: CALL метка (адрес) или имя процедуры
Логика:
PUSH CS
CS = dest_seg
PUSH IP
IP = dest_offset

JMP Безусловный переход

Флаги не меняются.
Команда: JMP метка (адрес)

RET Возврат из подпрограммы

Флаги не меняются.
Команда: RET число
Логика:
POP IP
POP CS
SP = SP + число (если оно имеется).

2.6. Команды условного перехода

Ни одна из команд условного перехода флаги не меняет. В качестве параметра в командах условного перехода указывается метка (ближняя ссылка – short_label). При вычислении адреса перехода по короткой ссылке, к текущему значению указателя команд (IP) прибавляется короткое целое число (диапазон изменения числа от -128 до +127) и, таким образом, вычисляется адрес следующей команды для выполнения.

JA Переход если выше (JNBE Переход если не ниже и не равно)

Условие перехода: (C = 0) and (Z = 0) .

JAE Переход если выше или равно (JNB не ниже, JNC нет переноса)

Условие перехода: C = 0

JB Переход если ниже (JNAE Переход если не выше и не равно)

Условие перехода: C = 1 .

JBE Переход если ниже или равно (JNA Переход если не выше)

Условие перехода: (C = 1) or (Z = 1) .

JC Переход если перенос

Условие перехода: C = 1 .

JCXZ Переход если CX = 0

Условие перехода: CX = 0 .

JE Переход если равно (JZ Переход если ноль)

Условие перехода: Z = 1 .

JG Переход если больше (JNLE переход если не меньше и не равно)

Условие перехода: (Z = 0) and (S = O) .

JGE Переход если больше или равно (JNL Переход если не меньше)

Условие перехода: S = O .

JL Переход если меньше (JNGE Переход если не больше и не равно)

Условие перехода: S <> O .

JLE Переход если меньше или равно (JNG Переход если не больше)

Условие перехода: (S <> O) or (Z = 1) .

JNE Переход если не равно (JNZ Переход если не ноль)

Условие перехода: Z = 0

JNO Переход если нет переполнения

Условие перехода: O = 0 .

JNP Переход если нечётно (JPO Переход если нечетно)

Условие перехода: P = 0 .

JNS Переход если положительный результат

Условие перехода: S = 0 .

JO Переход если есть переполнение

Условие перехода: O = 1 .

JP Переход если чётно (JPE Переход если четно)

Условие перехода: P = 1 .

JS Переход если отрицательный результат

Условие перехода: S = 1 .

LOOP Переход по счетчику

логика: CX = CX - 1
if (CX <> 0) JMP short-label .

LOOPNE Переход пока не равно (LOOPNZ Переход пока не ноль)

Логика: CX = CX - 1
if (CX <> 0) and (Z = 0) JMP short-label .

LOOPZ Переход пока ноль

Логика: CX = CX - 1
if (CX <> 0) and (Z = 1) JMP short-label .

2.7 Управление состоянием процессора

CLC Сброс флага переноса

Делает значение флаг переноса равным нулю (C = 0).

CLD Сброс флага направления

D = 0. Разрешает инкремент в командах обработки строк.

CMC Инвертирование флага переноса

C = not C

NOP Нет операции

Логика: нет.

STC Установка флага переноса

Делает значение флаг переноса равным единице (C = 1).

STD Установка флага направления

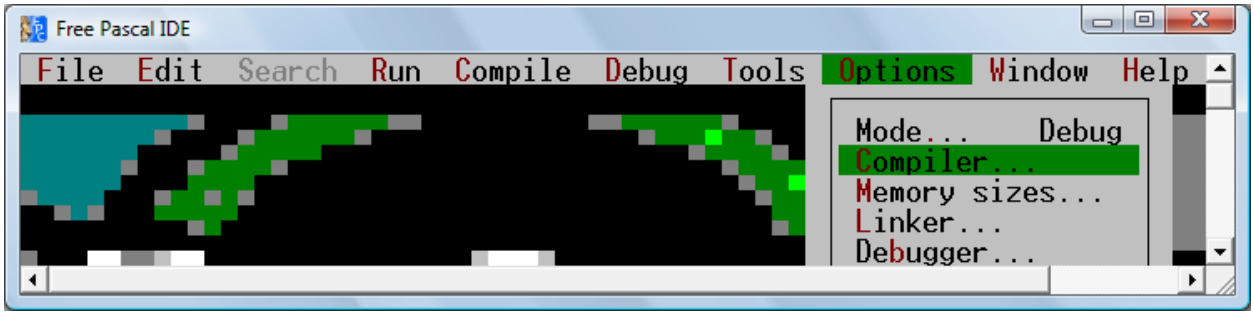
D = 1 (декремент в командах обработки строк)

3. РАБОТА В ИНТЕГРИРОВАННОЙ СРЕДЕ FREE PASCAL

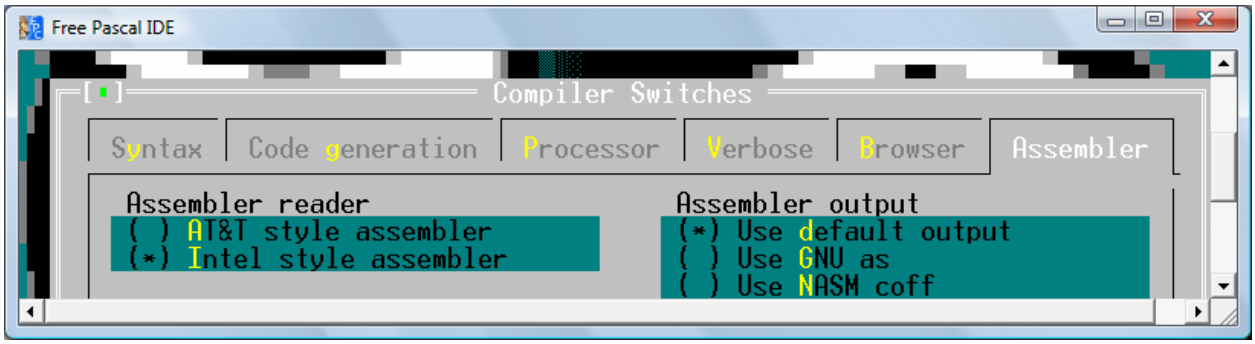
3.1. Настройка интегрированной среды.

Выбор типа стиля записи ассемблерного кода.

Для выбора стиля ассемблерного кода надо в меню «Options» выбрать настройку компилятора «Compiler…».

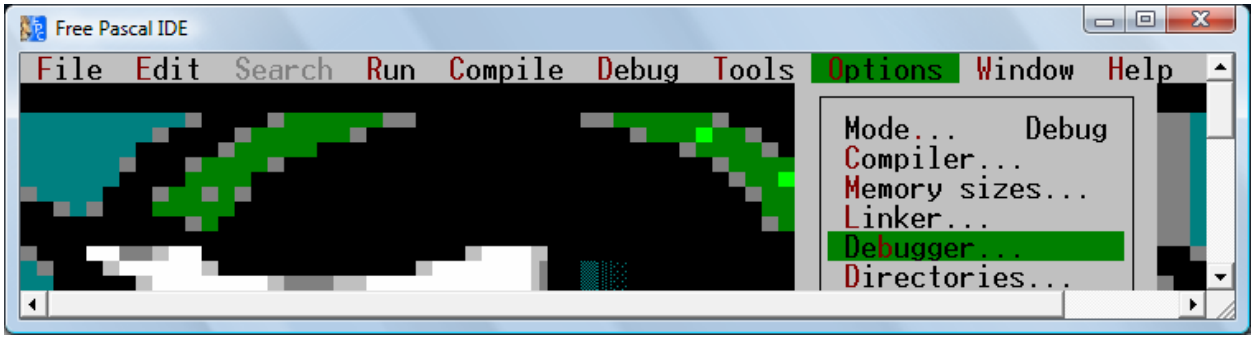


А потом в окне «Compiler Switches» выбрать вкладку «Assembler», а в ней в окне «Assembler reader» установить радио-кнопку «Intel style assembler».

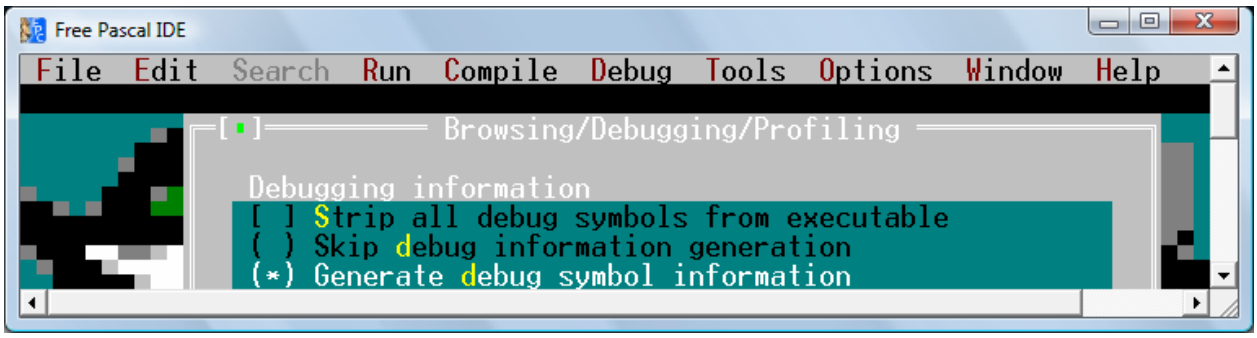


Установка режима работы отладчика

Для выбора стиля ассемблерного кода надо в меню «Options» выбрать настройку отладчика «Debugger…».

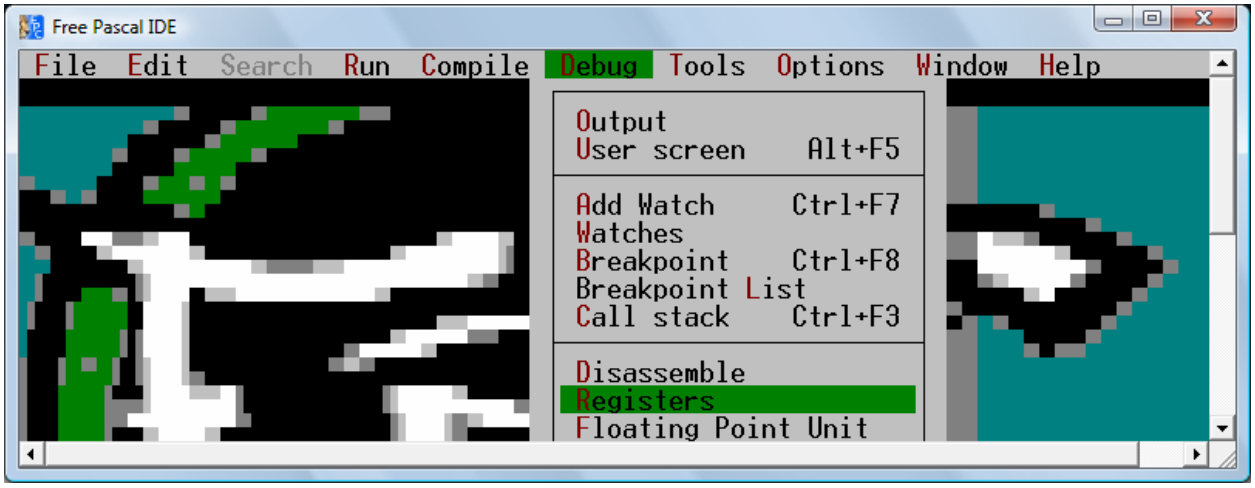


А потом в окне «Debugging information» установить радио-кнопку «Generate debug symbol information».

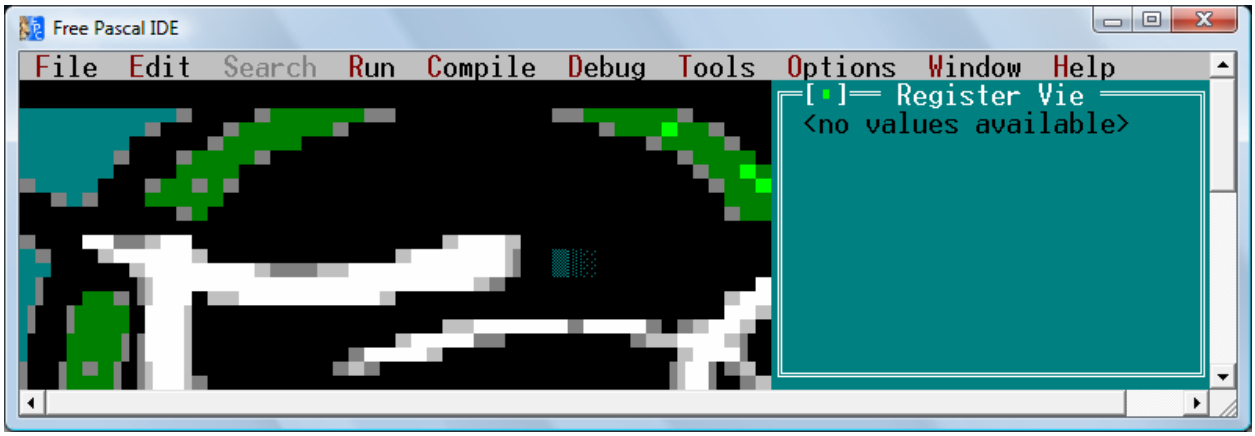


Просмотр содержимого регистров процессора

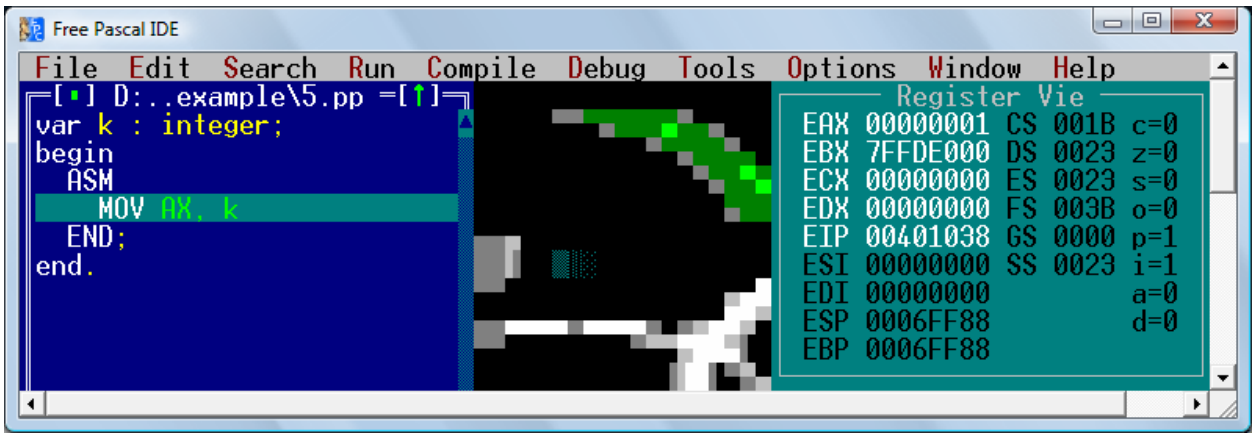
Просмотр содержимого регистров процессора проводится в отдельном окне «Register Vie», которое вызывается из меню «Debug».



До запуска программы на исполнение окно «Register Vie» пустое.

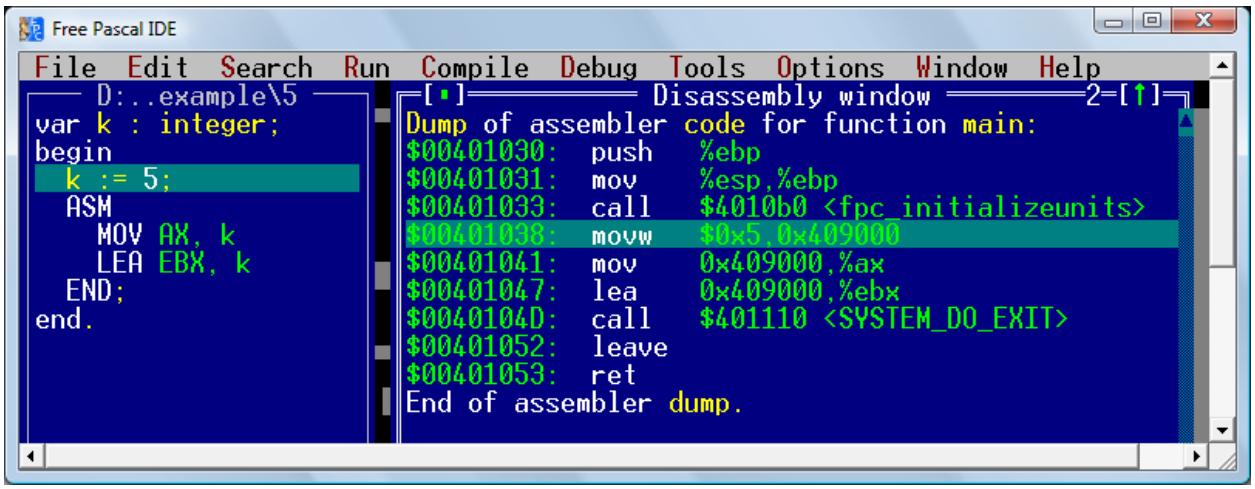


Чтобы увидеть содержимое регистров, надо выполнять программу по шагам. При этом в окне «Register Vie» будут отображаться 15 регистров. Числа, содержащиеся в регистрах, представлены в шестнадцатеричной системе счисления. Изменённые значения в регистрах выделяются цветом.



Просмотр программы в окне Disassembly window

Окно «Disassembly window» показывает так называемый дизассемблированный вид программы, то есть, как выглядит программа в оперативной памяти в кодах. Вызов окна производится также, как и окна просмотра содержимого регистров. Так же исходно окно будет пустое и будет заполнено только после запуска программы на исполнение. Например, может быть такой вид окна:



Как видно из рисунка, даже оператор присваивания языка Паскаль представлен в виде кодов и ассемблерной мнемоники.

В каждой строке дизассемблированного кода в первом столбце указывается адрес команды ассемблера. Например, текущая команда (присваивание переменной k значения 5) расположена по адресу $00401038. А следующая команда – по адресу $00401041. Нетрудно догадаться, что на текущую команду будет потрачено 9 байтов (числа шестнадцатеричные!). Что означает каждый из девяти байтов, можно узнать из описания системы команд. Второй столбец – это команда процессора в мнемонике ассемблера. Третий столбец – операнды.

Таким образом, просматривая дизассемблированный код можно увидеть, во что трансформируются операторы языка высокого уровня, какие ис-

пользуются для этого команды процессора, по каким адресам расположены переменные (видно, что переменная «k» находится по адресу 0x409000).

3.2. Включение ассемблерного кода в программу на Паскале.

Есть два способа включения ассемблерного кода в программу на языке Паскаль. Первый из них – использование ассемблерного блока (для наглядности ассемблерная часть выделена прописными буквами):

```
var
  k : Integer;
begin
   k := 3;
   ASM
      MOV  AX, k
   END;
end.
```

Второй способ – реализация процедуры или функции на ассемблере:

```
procedure Primer; ASSEMBLER;
ASM
   MOV AX, 7
END;
```

Внутри процедур или функций можно объявлять и использовать локальные переменные. В этом случае компилятор будет резервировать под них место в стеке.

```
procedure Primer; ASSEMBLER;
var perem : integer;
ASM
   MOV AX, 7
   MOV perem, AX
END;
```

4. ЛАБОРАТОРНЫЕ ЗАДАНИЯ

Лабораторными заданиями предусматривается изучение команд процессора, используя средства встроенного ассемблера и интегрированной среды Free Pascal. Отдельного задания на команды управления состоянием процессора нет. Они могут быть использованы при изучении других команд.

Отчёт по каждому заданию оформляется в виде программы на языке Паскаль со встроенными блоками на ассемблере. Программа по каждому заданию должна содержать в виде комментария сведения об авторе, заголовок и краткие комментарии. Например:

```
// Лабораторное задание № 1
// Изучение команд пересылки данных
// Выполнил студент гр. ИСТд-21 Великий А.А.
```

Лабораторное задание считается сданным при выполнении двух условий: первое – наличие правильно оформленного отчёта, второе – выполнение небольшой контрольной задачи (правила оформления отчёта и примерные варианты задач приведены в описании каждого из лабораторных заданий). Задачи должны выполняться на занятиях.

4.1. Изучение команд пересылки данных

Изучение команд пересылки данных – одно из наиболее объемных заданий. Оно предусматривает изучение логики работы каждой из перечисленных ранее команд.

Кроме изучения логики работы команд пересылки данных, при выполнении данного задания надо на примере одной из команд надо рассмотреть все режимы адресации. Лучше всего для этих целей подходит команда MOV. Также надо будет указать результат каждой операции. Направление пересылки можно указать стрелкой. Для каждого из режимов адресации укажите в комментарии, какие регистры можно (или какие регистры нельзя) использовать. Например:

```
var a : integer;
begin
  a := 12;
  asm
    // РЕЖИМЫ АДРЕСАЦИИ
    // непосредственный режим адресации, в качестве приёмника
    // нельзя использовать регистры …
    MOV  AX, 23 // источник – непосредственная, приёмник – регистровая, (23 -> AX)
    MOV  AX, a   // источник – прямая, приёмник – регистровая (a -> AX = 12)
    //косвенно-регистровая адресация (можно использовать регистры …)
    LEA   EBX, a // адрес переменной a -> EBX (EBX = 00409000h)
    MOV  CX, [EBX] // из ячейки памяти с адресом EBX = 00409000h -> CX = 12
  end;
end.
```

При использовании команд получения адреса (например, LEA) не забывайте показать, как этот адрес используется (см. следующую за командой LEA строку). При этом покажите, какие числа пересылаются.

При рассмотрении команд LES и LDS обратите внимание на существенное отличие этих команд от команды LEA. Команда LEA определяет адрес размещения самой переменной языка Паскаль, а команды LES и LDS - заносят в регистры содержимое переменной типа указатель. Чтобы не потерять значение сегмента данных DS, перед выполнением команды LDS его содержимое сохраняется в стеке, а после выполнения команды восстанавливается из стека:

```
var p : ^integer;
begin
   getmem(p,2);
   p^ := 7;
  ASM
      LES    EBX, p
      MOV   CX, [EBX] // p^ = 7 -> CX
      PUSH DS
      LES    EBX, p
      MOV   CX, [EBX]
      POP   DS
  END;
end.
```

При реализации в командах косвенно-регистрового режима адресации со смещением и выполнении команды XLAT, используйте массивы данных с обязательным указанием, того к какому элементу массива идёт обращение. Содержимое массивов должно быть подготовлено на языке Паскаль и указано в комментарии, что там находится. Например:

```
var
  M : array[0..15] of char;
  C : char;
  i : byte;
begin
  // Заполнить массив M символами 16-ричных цифр от 0 до F
  C := '0';
  for i := 0 to 9 do     begin       M[i] := C; inc(C);     end;
  C := 'A';
  for i := 10 to 15 do     begin       M[i] := C; inc(C);     end;
  // Извлечь из массива символ с номером 13 и поместить в переменную C
  ASM
    MOV    AL, 13
    LEA     EBX, M  // 0 1 2 3 4 5 6 7 8 9 A B C D E F
    XLAT               //  выбирается элемент ---^
    MOV    C, AL     // AL -> C = 'D'
  END;
end.
```

Примерные варианты контрольных задач:

1. Обменять значения в переменных языка Паскаль x : integer и y : ^integer.
2. Обменять значения в переменных языка Паскаль x[4] и y^[3], при выборке значения из массива y^ используйте команду XLAT.
3. Обменять значения в переменных языка Паскаль x[4] и y^[3]. Используйте команды PUSH и POP для временного хранения значений элементов массива.
4. Определите, сколько байт требуется на запись в оперативной памяти команды LEA EBX,M и, какие числа записаны в этих байтах.
5. Используя команды пересылок, покажите, как работает команда CMC.
6. Содержимое регистра флагов поместите в переменную x : integer.
7. Обменять значения в переменных языка Паскаль x : integer и y : ^integer. Обязательно использовать команду XCHG.

4.2. Изучение арифметических команд

При выполнении задания необходимо обратить внимание на формат получаемого результата, как изменяются флаги при выполнении арифметических команд в зависимости от исходных данных. Рассмотреть отличия команд INC и DEC от команд сложения ADD и вычитания SUB (состояние флага C). Используя окно дизассемблера, посмотреть, во что транслируются арифметические операции языка Паскаль.

А также выполнить следующие требования к отчёту:

1. Все арифметические команды должны содержать исходные данные рядом с командой.
2. Команды умножения и деления покажите над десятичными числами.
3. Команды сложения и вычитания покажите с такими исходными данными, чтобы изменялись флаги переполнения и переноса (на каждую команду два примера, на изменение этих флагов по отдельности), дайте в отчете комментарий, поясняющий результат.
4. Результат выполнения команды должен присутствовать в виде комментария.

Отличия назначения и использования флагов переполнения и переноса можно рассмотреть на следующем примере:

```
begin
  ASM
    MOV AL, 01111111B // число со знаком
    ADD AL, 00000001B // переполнение 7 разрядов
    MOV AL, 11111111B // число без знака
    ADD AL, 00000001B // перенос во флаг C
  END;
end.
```

Примерные варианты контрольных задач:

1. Реализовать сложение двух 64-разрядных чисел.
2. Реализовать вычитание двух 64-разрядных чисел.
3. Реализовать вычитание двух 64-разрядных чисел, не используя команду SUB.
4. Показать на примере реализацию команд умножения и деления командами 32-разрядного процессора (формат посмотреть в окне дизассемблера).

4.3. Изучение логических команд и команд сдвигов

При выполнении задания требуется все исходные данные и результат представлять в двоичном виде. Кроме демонстрации работы логических команд, требуется особо выделить и продемонстрировать: отличия команды NOT от команды NEG, отличия команд SHR и SAR, отличия команд SUB и TEST, отличия циклических сдвигов и циклических сдвигов через флаг C.

Некоторые, часто используемые приёмы работы с использованием логических команд:

1. Очистка содержимого регистра командой «исключающее или», в которой и источник и приёмник – один и тот же регистр (XOR CX,CX).
2. Проверка присутствия бита на заданной позиции с помощью маски, например, есть ли в регистре AX единица в 5-м бите, проводится командой TEST (TEST AX,100000B).
3. Использование маски для очистки одного бита (или нескольких битов) командой «логическое и». На очищаемые позиции в маске устанавливается 0, а в остальных единицы, например, если в AL надо 5-й бит установить равным нулю, то пишется команда: AND AL,11011111B.
4. Аналогично для установки бита в конкретной позиции, используется команда «логическое или» (AND AL,00100000B).
5. Проверку содержимого крайних битов осуществляют сдвигом их во флаг переноса C.
6. Умножение (деление) на число, равное степени числа 2, делают арифметическим сдвигом влево (вправо).

Примерные варианты контрольных задач:

1. Реализовать циклический сдвиг влево 32-разрядного числа, оперируя только 16-разрядными регистрами.
2. Реализовать циклический сдвиг вправо 32-разрядного числа, оперируя только 16-разрядными регистрами.
3. Реализовать умножение на 7 (или 15, или 17, или 33) используя команды сдвигов и (только один раз) сложение или вычитание.
4. Реализовать умножение числа 2000000099 на 10.

4.4. Изучение команд обработки блоков данных. Цикл LOOP.

Во время изучения логики работы команд обработки блоков данных во избежание ошибок реализуйте программу сначала без циклов, чтобы можно было бы проконтролировать процесс выполнения по шагам. Например, для решения задачи заполнения переменной типа string заданным количеством одинаковых символов, запишите несколько строк с командой STOSB, и только убедившись в правильности алгоритма, реализуйте цикл (REP STOSB):

```
var   S : string;  C : char;  N : byte;
begin
  S := ''; // строка для заполнения
  C := 'A'; // символ для заполнения строки
  N := 3; // количество символов в строке
  ASM
    LEA EDI, S      // адрес строки
    XOR CX,CX    // очистка счётчика CX = 0
    MOV CL,N      // в счётчик занести количество повторений
    MOV AL,C      // символ для заполнения поместить в AL
    MOV [EDI],CL // заполнить S[0] – количество символов в строке
    INC EDI          // символы записывать, начиная с S[1]
    CLD               // установить инкремент адреса
    STOSB          // записать три раза по одному символу
    STOSB
    STOSB
  END;
end.
```

При необходимости использования цикла в цикле, можно использовать цикл LOOP, но при выполнении вложенного цикла не забывайте сохранять значение регистра CX (счётчик внешнего цикла) в стеке:

```
Label L1,L2;
begin
  ASM
      MOV CX,2
L1:    PUSH CX
       MOV CX,3
L2:      NOP
       LOOP L2
       POP CX
      LOOP L1
  END;
end.
```

Примерные варианты контрольных задач:

1. Найти в строке позицию заданного символа.
2. Определить, есть ли в двух строках одинаковые символы на одинаковых позициях.
3. Найти позицию, на которой две строки символов отличаются.

4. Удалить из строки заданный символ.
5. Удалить из строки символ на заданной позиции.

4.5. Изучение команд условного перехода

Это задание оказывается объёмным, но простым по исполнению. Для каждого условного перехода надо сделать запись из двух примеров: когда условие перехода выполняется, и, когда условие перехода не выполняется. Например:

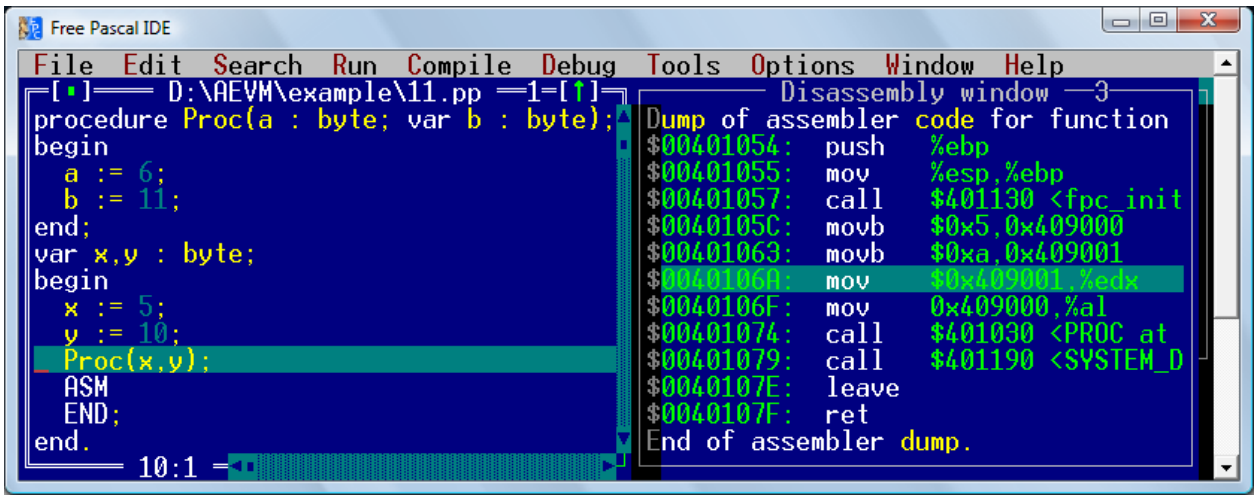
```
Label L1,L2;
begin
  ASM
      MOV AL,67
      CMP AL,34    // Содержимое регистра AL>34 ?
      JA  L1       // переход на метку L1, т.к. условие выполнено
      NOP          // команда NOP будет пропущена
L1:   CMP AL,84    // Содержимое регистра AL > 84 ?
      JA  L2       //перехода на метку L2 нет, т.к. условие не выполнено
      NOP          // команда NOP будет выполнена
L2:   NOP
  END;
end.
```

Примерные варианты контрольных задач:

1. Реализовать на языке ассемблера вычисление выражения, записанного на языке Паскаль: if (x>=5)and(x<=7) then x:=1 else x:=2;
2. Реализовать на языке ассемблера вычисление выражения, записанного на языке Паскаль: if (x<5)or(x>7) then x:=1 else x:=2;

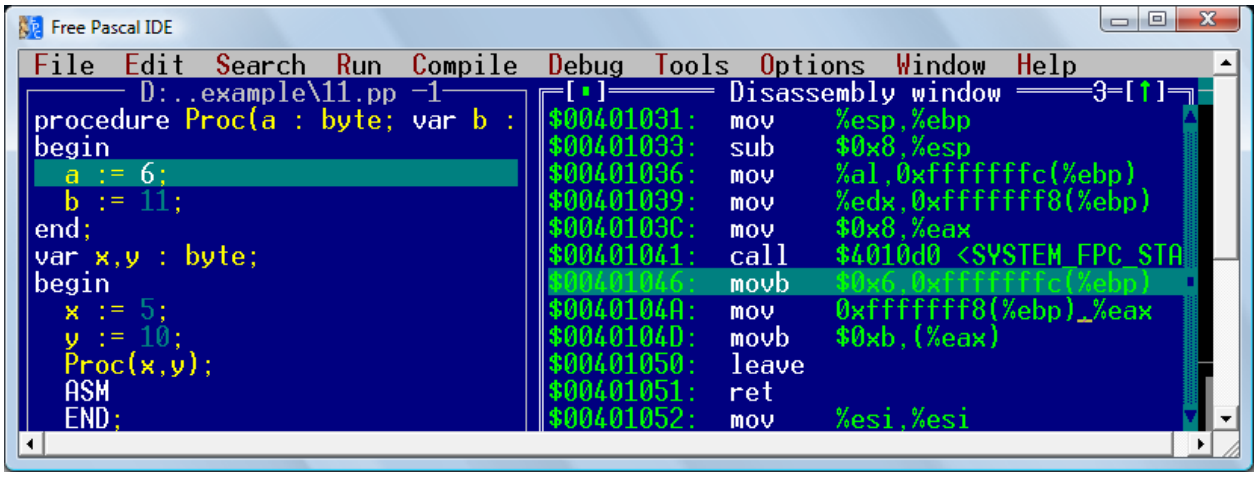
4.6. Изучение команд передачи управления.

При выполнении задания кроме самих команд необходимо рассмотреть вопросы передачи параметров переменных и параметров значений в процедурах языка Паскаль. Надо показать, что будет размещаться в стеке: само значение или адрес переменной. Это можно сделать, рассмотрев в окне дизассемблера программу:



В этом примере в окне дизассемблера видно, что перед вызовом процедуры Proc в регистр EDX передаётся число 0x409001, являющееся адресом переменной «y». А в регистр AL пересылается значение переменной «x». Это соответствует тому, что переменная «x» подставляется на место формального параметра значения «a», а переменная – на место формального параметра переменной «b».

Когда начинает работу процедура, выполняется несколько действий, показанных в следующем окне дизассемблера:



До выделенной строки в стек записываются содержимое регистров AL и EDX. Далее выполняется библиотечная подпрограмма языка Паскаль. И только потом действия, записанные в самой процедуре. Первое присваивание (a := 6) транслируется в пересылку числа 6 в стек. Второе присваивание (b := 11) делается в два этапа.

В качестве вариантов контрольных задач предлагается реализовать на языке ассемблера одну из функций работы со строками языка Паскаль или Си (Pos, Insert, Delete, strchr, strchrr, strstr и т.д.).

СПИСОК ЛИТЕРАТУРЫ

Цилькер Б.Я. Организация ЭВМ и систем: Учебник для вузов. / Б. Я. Цилькер, С.А. Орлов. – СПб.: Питер, 2004. – 668 с.

Абель П. Язык ассемблера для IBM PC и программирования / Пер. с англ. Ю.В.Сальникова. – М.: Высш. шк., 1992. – 447 с.

ОГЛАВЛЕНИЕ

Учебное издание

Архитектура ЭВМ
Методические материалы

Авторы: ДОКТОРОВ Александр Евгеньевич
ДОКТОРОВА Елена Анатольевна

Редактор _____
Подписано в печать _____. Формат _____
Бумага писчая. Печать трафаретная. Усл. печ. л. _____.
Уч.-изд. л. _____. Тираж 50 экз. Заказ _____.
Ульяновский государственный технический университет
432027, Ульяновск, Сев. Венец, 32.
Типография УлГТУ, 432027, Ульяновск, Сев. Венец, 32.