Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Муромский институт (филиал)
Государственного образовательного учреждения
высшего профессионального образования
«Владимирский государственный университет»

Программирование на языке ассемблера

Методические указания к лабораторному практикуму

Часть 1

Составители:
Бейлекчи Д.В.
Калинкина Н.Е.

Муром
2007

УДК 681.3.06
ББК 32.973 – 018.1
П 78

Рецензент:

кандидат физико-математических наук,
доцент кафедры электроники и вычислительной техники
Муромского института (филиала) Владимирского государственного университета
Кулигин Михаил Николаевич

Печатается по решению редакционно-издательского совета
Муромского института

П 78
Ч.1 Программирование на языке ассемблера: методические указания к лабораторному практикуму. В 2 ч. Ч. 1. / Сост. - Бейлекчи Д.В., Калинкина Н.Е. – Муром: Изд. - ИПЦ МИ ВлГУ, 2007. - 60 с.: ил., библиогр. 10 назв.

В методических указаниях приведено описание восьми лабораторных работ по курсу «Архитектура микропроцессора и программирование на языке ассемблера». Первая часть содержит описание лабораторных работ № 1-3, вторая - № 4-8. Каждая лабораторная работа содержит краткие теоретические сведения и рассчитана на четыре академических часа самостоятельной работы. После выполнения всего курса лабораторных работ студент должен приобрести базовые знания и навык в программировании на языке ассемблера.

Курс лабораторных работ рассчитан на студентов специальности 230101.65 «Вычислительные машины, комплексы, системы и сети», 230105.65 «Программное обеспечение вычислительной техники и автоматизированных систем», но может быть полезен и тем, кто самостоятельно изучает язык ассемблера.

УДК 681.3.06
ББК 32.973 – 018.1

Предисловие

Результат работы самого лучшего оптимизирующего транслятора с языка высокого уровня не может конкурировать по скорости или компактности с кодом, написанным опытным программистом на языке ассемблера. Поэтому на языке ассемблера пишут, когда скорость работы программы или занимаемая ею память имеют решающее значение. Язык ассемблера незаменим для получения полного доступа к некоторым специфическим функциям процессора или периферийных устройств.

Язык ассемблера предоставляет возможность изучения процессора, его возможностей и ограничений, что позволяет лучше понять принципы функционирования как аппаратных, так и программных систем.

Курс лабораторных работ рассчитан на студентов специальности 230101.65 «Вычислительные машины, комплексы, системы и сети», 230105.65 «Программное обеспечение вычислительной техники и автоматизированных систем», но может быть полезен и тем, кто самостоятельно изучает язык ассемблера.

В методических указаниях приведено описание восьми лабораторных работ по курсу «Архитектура микропроцессора и программирование на языке ассемблера». Первая часть содержит описание лабораторных работ № 1-3. Каждая лабораторная работа содержит краткие теоретические сведения и рассчитана на четыре академических часа самостоятельной работы. После выполнения всего курса лабораторных работ студент должен приобрести базовые знания и навык программирования на языке ассемблера.

Лабораторная работа № 1

ЗНАКОМСТВО С ПРОГРАММАМИ В МАШИННЫХ КОДАХ

Цель работы: Изучение структуры машинных команд и методов работы с шестнадцатеричным редактором.

ДОМАШНЯЯ ПОДГОТОВКА

- Изучить программную модель микропроцессора i8086:
  - Программно-доступные регистры;
  - Организацию памяти;
  - Формат команд;
  - Режимы адресации данных i8086;
- Подготовить ответы на контрольные вопросы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Программная модель процессора Intel 8086

Организация памяти

Для хранения программы и данных в микропроцессоре (МП) intel 8086 (i8086) используются одно пространство памяти. Такая организация получила название *архитектуры Джона фон Нейман.*

В i8086 выделяется адресное пространство небольшого объема, которое называется набором программно-доступных *регистров МП*. В отличии от памяти программ и памяти данных регистры располагаются внутри МП, что обеспечивает быстрый доступ к информации, хранящейся в них.

Кроме регистров, в МП существует специальная область называемая стеком. *Стек* – это область памяти, специально выделяемая для временного хранения данных программы. Главная функция стека это организация подпрограмм: в стеке хранятся адреса возврата из подпрограмм (адрес с которого необходимо продолжить программу после

завершения подпрограммы), а также параметры, передаваемые в подпрограмму. Принцип работы стека позволяет организовывать вложенные вызовы подпрограмм.

В МП i8086 используется *сегментная* модель памяти. При такой организации основная память (данных, программ и стека) разделяется на блоки – *сегменты*, которые поддерживаются на аппаратном уровне. Сегментная организация обеспечивает существование нескольких независимых адресных пространств как в пределах одной задачи (программы), так и в системе в целом.

При сегментной организации внутри программы используются логические (виртуальные) адреса, которые состоят из адреса сегмента и смещения относительно начала сегмента. Логический адрес принято записывать в формате «сегмент : смещение». Вторая часть адреса, смещение, называется также эффективным (исполнительным) адресом.

Поскольку для адресации памяти процессор использует 16-разрядные адресные регистры, то это обеспечивает ему доступ к 2^{16}=65536 байт или 64 Кб основной памяти. Однако адресная шина, соединяющая процессор i8086 и память 20-разрядная и МП генерирует 20-битовые физические адреса. (Под физическим адресом понимается адрес памяти, выдаваемый на шину адреса микропроцессора.) Таким образом, МП i8086 может адресовать 2^{20} (1 Мб) памяти.

Рассмотрим порядок формирования физического адреса из логического. В МП хранятся 16-разрядные базовые адреса используемых сегментов. Микропроцессор объединяет 16-разрядный базовый адрес и 16-разрядный исполнительный адрес следующим образом: он расширяет содержимое сегментного регистра (базовый адрес) четырьмя нулевыми битами (в младших разрядах), делая его 20-разрядным (полный адрес сегмента) и прибавляет смещение (исполнительный адрес) (рис. 1).

Эта специальная процедура пересчёта адресов поддерживается на аппаратном уровне. Полученный 20-разрядный результат является физическим (абсолютным) адресом ячейки памяти.

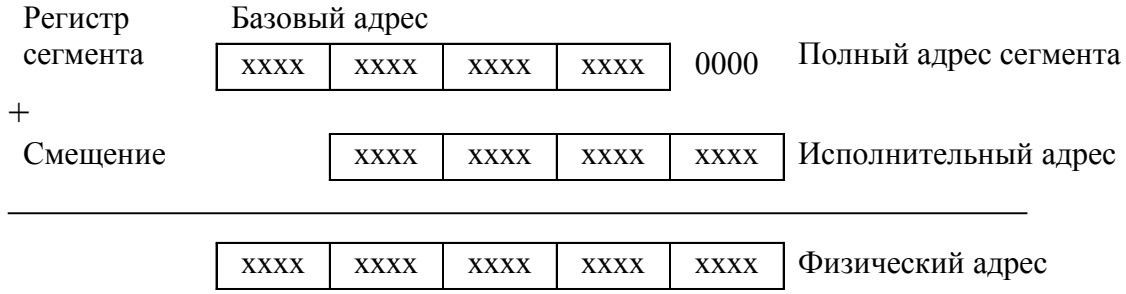


Рис. 1. Формирование физического адреса.

Регистры

В программной модели процессора i8086 имеется 14 шестнадцати-разрядных регистров, которые используются для управления исполнением команд, адресации и выполнения арифметических операций.

Регистры общего назначения AX, BX, CX, DX, SI, DI, BP, SP единообразно участвуют в командах манипуляции данными. При этом с каждым регистром связана некоторая специальная функция, что нашло отражение в названиях регистров.

Регистры данных. У 16-разрядных регистров AX, BX, CX, DX отдельно адресуются старшие (High) и младшие (Low) байты: AH и AL в AX, BH и BL в BX, CH и CL в CX, DH и DL в DX, таким образом регистры данных можно рассматривать как четыре 16-разрядных или восемь 8-разрядных регистров. Перечислим наиболее характерные функции каждого них:

– AX (AL) – аккумулятор (accumulator) служит для хранения промежуточных данных; многие команды оперируют данными в аккумуляторе несколько быстрее, чем в других регистрах;
– BX – базовый регистр (base) применяется для указания базового (начального) адреса объекта данных в памяти;

– CX (CL) – регистр-счётчик (counter) участвует в качестве счётчика в некоторых командах, которые производят повторяющиеся операции;
– DX – регистр данных (data) используется главным образом для хранения промежуточных данных в качестве расширителя аккумулятора.

Регистры указателей и индексов. В микропроцессоре существуют шесть 16-разрядных регистров, которые могут принимать участие в адресации операндов. Один из них относиться к регистрам данных – это регистр базы BX. Кроме BX в явной адресации участвуют указатель базы BP (base pointer), индекс источника SI (source index) и индекс приемника DI (destination index). Основное назначение этих регистров – хранение значений, используемых при формировании адресов операндов. Разнообразные способы сочетания в командах этих регистров и других величин называются способами (режимами) адресации. Эти регистры можно также использовать и для временного хранения данных.

Два оставшихся регистра SP (stack pointer) и IP (instruction pointer) – используются при адресации неявно.

Регистр указателя стека. Указатель стека SP – это 16-разрядный регистр, который определяет смещение текущей вершины стека. Указатель стека SP вместе с сегментным регистром стека SS используются микропроцессором для формирования физического адреса вершины стека. Регистровая пара SS:SP всегда указывает на текущею вершину стека. Стек растет в направлении младших адресов памяти, т.е. перед тем как слово помещается в стек, содержимое SP уменьшается на 2, после того как слово извлекается из стека, микропроцессор увеличивает содержимое регистра SP на 2.

Регистр указателя команд. Микропроцессор использует регистр указателя команд IP совместно с регистром CS для формирования 20-битового физического адреса очередной выполняемой команды, при

этом регистр CS определяет сегмент выполняемой программы, а IP – смещение от начала сегмента. По мере того, как микропроцессор загружает команду из памяти и выполняет её, регистр IP увеличивается на длину команды в байтах. Для непосредственного изменения содержимого регистра IP служат команды перехода.

Регистры сегментов. Имеются четыре регистра сегментов, с помощью которых память можно организовать в виде совокупности четырёх различных сегментов (до 64 Кбайт). Эти 4 различные области памяти могут располагаться практически в любом месте физической памяти. Поскольку местоположение каждого сегмента определяется только содержимым соответствующего регистра сегмента, для реорганизации памяти достаточно всего лишь, изменить это содержимое.

– CS – сегментный регистр кода (code segment) указывает на сегмент, содержащий текущую исполняемую программу. Пара CS:IP определяет адрес команды, которая должна быть выбрана для выполнения;
– DS – сегментный регистр данных (data segment) указывает на текущий сегмент данных;
– SS – сегментный регистр стека (stack segment) указывает на текущий сегмент стека;
– ES – дополнительный сегментный регистр (extended segment) указывает на дополнительную область памяти, используемую для хранения данных.

Регистр флагов. Каждый бит (флаг) 16-разрядного регистра флагов (признаков) имеет свое значение. Некоторые из этих бит, *флаги состояния*, отражают особенности результата команд обработки данных. Другие биты, *флаги управления*, показывают текущее состояние микропроцессора. Микропроцессор 8086 использует только 9 флагов, остальные зарезервированы.

Все флаги младшего байта регистра флагов устанавливаются арифметическими или логическими командами микропроцессора и могут быть опрошены командами условного перехода для изменения порядка выполнения программы. За исключением флага переполнения, все флаги старшего байта отражают состояние микропроцессора и влияют на характер выполнения программы. Флаги состояния и CF устанавливаются и сбрасываются специально предназначенными для этого командами.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	×	×	×	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

Биты регистра флагов имеют следующее назначение:

× – зарезервированные биты.

OF – флаг переполнения (overflow) равен 1, если возникает арифметическое переполнение, т.е. когда результат превышает диапазон представимых чисел;

DF – флаг направления (direction) устанавливается в 1 для автоматического декремента и в 0 для автоинкремента индексных регистров после выполнения операции над строками;

IF – флаг прерывания (interrupt). Если IF = 1, то прерывания разрешены. Если же IF = 0, то распознаются лишь немаскируемые прерывания;

TF – флаг трассировки (trace). Если TF равен 1, то процессор переводится в режим покомандной работы, т.е. после выполнения каждой команды генерируется внутреннее прерывание с переходом к отладчику;

SF – флаг знака (size) равен 1, когда старший бит результата равен 1. Т.о. SF=0 для положительных чисел, и 1 для отрицательных;

ZF – флаг нуля (zero) равен 1, если результат равен нулю;

AF – флаг вспомогательного переноса (auxiliary carry). Этот флаг устанавливается в 1, если арифметическая операция вызвала перенос или заём из младшей тетрады;

PF – флаг чётности (parity) устанавливается в 1, если младшие 8 бит результата содержат чётное число единичных бит;

CF – флаг переноса (carry) устанавливается в 1, если имеет место перенос или заём из старшего бита результата.

Режимы адресации

Режимом адресации называется способ указания адреса операнда необходимого для выполнения операции. В зависимости от спецификаций и местоположения источников образования полного (абсолютного) адреса в языке ассемблера различают следующие способы адресации операндов: неявная, регистровая, непосредственная, косвенная (прямая, базовая, индексная, базово-индексная).

При неявной адресации операнд или адрес операнда находится в строго определённом месте (обычно в регистре) и поэтому в команде не указывается. Например, в команде LODSB операндами являются регистры AL, SI и флаг DF (в AL записывается байт с адреса DS:SI, затем смещение в SI увеличивается или уменьшается на 1 в зависимости от флага DF).

При регистровой адресации операнд находится в регистре. Например, в команде MOV **AX**,**BX** для обоих операндов используется регистровая адресация.

При непосредственной адресации операнд находится в самой команде, т.е. хранится вместе с командой в сегменте кода, поэтому изменить операнд нельзя и им может быть только константа. Например, в команде MOV AX, **200** для второго операнда используется непосредственная адресация.

При использовании косвенной адресации абсолютный адрес формируется исходя из базового адреса в одном из сегментных регистров и смещения:

```
MOV     AX, [20]            ; База – в DS, смещение в команде – 20
```

```
MOV     AL, [BP +10]        ; База – в SS, смещение – сумма = BP +10
MOV     AX, ES:[SI]         ; База – в ES, смещение – в SI
```

Виды косвенной адресации представлены в таблице:

Косвенная адресация	Формат	Сегмент по умолчанию	Примеры
Прямая	метка, смещение (ofs)	DS DS	Mov [lab], AX Mov [1234h], AX
Базовая	[BX+ofs] [BP+ofs]	DS SS	Mov [BX+1], AX Mov [BP+lab], AX
Индексная	[DI+ofs] [SI+ofs]	DS DS	Mov [DI+12h], AX Mov [SI+lab], AX

Базово-Индексная	[BX+SI+ofs]
	[BX+DI+ofs]
	[BP+SI+ofs]
	[BP+DI+ofs]
		DS
		DS
		SS
		SS
			Mov [BX+SI-12h], AX
			Mov [BX+DI], AX
			Mov [BP+SI], AX
			Mov [BP+DI+lab], AX

При косвенной прямой адресации адрес (16-разрядное смещение) операнда находится в команде. Например, в команде MOV AX,[**100h**] для второго операнда используется прямая адресация, т.е. данные находятся в памяти по адресу DS:100h (сегментный регистр DS используется по умолчанию для адресации данных).

В случае применения косвенной базовой адресации исполнительный адрес является суммой значения смещения и содержимого регистра BP или BX, например:

```
MOV     AX,[BP+6]           ; База – SS, смещение – BP+6
MOV     DX,[BX][8]          ; База – DS, смещение – BX+8
```

При косвенной индексной адресации исполнительный адрес определяется как сумма значения указанного смещения и содержимого регистра SI или DI так же, как и при базовой адресации, например:

```
MOV     DX,[SI+5]           ;База – DS, смещение – SI+5
```

Косвенная базово-индексная адресация подразумевает использование для вычисления исполнительного адреса суммы содержимого базового регистра и индексного регистра, а также смещения, находящегося в операторе, например:

```
MOV     BX,[BP][SI]         ; База – SS, смещение – BP+SI
MOV     ES:[BX+DI+6],AX     ; База – ES, смещение – BX+DI+6
```

Формат команд i8086

Машинные команды i8086 состоят из необязательных префиксов, одного или двух байт главного кода операции, возможного спецификатора адреса ModR/M, содержащего байт, определяющий форму адреса и, если требуется, смещения в команде и поля непосредственных данных (рис. 2).

Префикс команды	Префикс замены сегмента	КОП	ModR/M	Смещение в команде	Непосредственный операнд
0 или 1 байт	0 или 1 байт	1 или 2 байта	0 или 1 байт	0,1,2 или 4 Байта	0,1,2 или 4 байта

Рис. 2. Общий формат команд МП i8086.

Поясним назначение полей машинной команды.

Префиксы. Необязательные элементы. В памяти префиксы предшествуют команде. Назначение префиксов – модифицировать операцию, выполняемую командой.

Префиксы команд:

F3	REP	F0	LOCK
F3	REPZ	F2	REPNZ

Префиксы замены сегментов:

2E	CS	3E	DS
36	SS	26	ES

Префикс замены сегмента в явной форме указывает, какой сегментный регистр используется в данной команде для адресации стека или данных. Префикс отменяет выбор сегментного регистра по умолчанию.

Код операции (КОП). Обязательный элемент, описывающий операцию, выполняемую командой. Следующие поля машинной команды определяют местоположение операндов, участвующих в операции, и особенности их использования.

Байт режима адресации (ModR/M) определяет используемую форму адреса операндов и состоит из трёх полей:

7	6	5	4	3	2	1	0
MOD		REG/КОП			R/M		

Поле MOD байта режима адресации ModR/M объединяется с полем R/M образуя 32 возможных значения: 8 регистров и 24 варианта режимов адресации. Например, если mod=01, то смещение в команде присутствует и занимает 1 байт. Если mod=11, то операндов в памяти нет, они находятся в регистрах.

Поле REG/КОП определяет либо номер регистра, либо дополнительные 3 бита кода операции.

Таблица 1. Варианты адресации в байте ModR/M.

R/m	mod = 00	mod = 01	mod = 10	mod = 11
000	[BX+SI]	[BX+SI+смещ8]	[BX+SI+смещ16]	AL / AX / ES
001	[BX+DI]	[BX+DI+смещ8]	[BX+DI+смещ16]	CL / CX / CS
010	[BP+SI]	[BP+SI+смещ8]	[BP+SI+смещ16]	DL / DX / SS
011	[BP+DI]	[BP+DI+смещ8]	[BP+DI+смещ16]	BL / BX / DS
100	[SI]	[SI+смещ8]	[SI+смещ16]	AH / SP
101	[DI]	[DI+смещ8]	[DI+смещ16]	CH / BP
110	смещ16	[BP+смещ8]	[BP+смещ16]	DH / SI
111	[BX]	[BX+смещ8]	[BX+смещ16]	BH / DI

Примечания:

смещ8 – означает 8 битное смещение после байта ModR/M;

смещ16 – означает 16 битное смещение после байта ModR/M.

Справочник машинных команд i8086 приведен в ПРИЛОЖЕНИИ 1.2.

Столбец **кода операции** в справочнике содержит код операции, формируемый для каждой формы команды. При описании команд используются следующие обозначения:

/цифра – показывает, что байт ModR/M кодирует только операнд r/m, а поле REG/КОП содержит цифру, являющуюся расширением кода операции.

+rb, +rw – код регистра (см. таблицу 1), который прибавляется к базовому коду операции с образованием одного байта кода операции.

Смысл поля R/M определяется КОП команды. Поле R/M может определять регистр как место нахождения операнда или служит частью кодирования режима адресации совместно с полем MOD.

В таблице 1 приведены значения полей байта ModR/M для различных вариантов адресации.

Рассмотрим примеры построения машинного кода команд.

Пример 1.

Команда	Режимы адресации аргументов	Шаблон из справочника	
Mov ah,bl	Mov r8,r8	Mov r8,r/m8	8A
Mov ah,bl	Mov r8,r8	Mov r/m8,r8	88

Воспользуемся шаблоном из первой строки. Как видим, в шаблоне нет конкретных аргументов, значит, нужен байт ModR/M. Заполняем его. Т.к. все аргументы – регистры, то mod = 11; Т.к. в шаблоне второй аргумент

r/m, то код второго аргумента в команде (bl=3) записываем в поле R/M, соответственно код первого аргумента (ah=4) запишем в поле REG/КОП:

1	1	1	0	0	0	1	1
MOD		REG/КОП			R/M		

Получили ModR/M = E3

Полный код команды = 8A E3

Но если посмотреть справочник, то есть ещё один шаблон, по которому можно построить код данной команды: Mov r/m8, r8 = 88. В этом случае в R/M запишем код ah, а в REG/КОП – код bl.

1	1	0	1	1	1	0	0
MOD		REG/КОП			R/M		

Получим ModR/M = DC

Полный код команды = 88 DC

Пример 2.

Команда	Режимы адресации аргументов	Шаблон из справочника
Mov byte ptr [bx+6], 24h	Mov m8,im8	Mov r/m8,im8 C6

Заполняем байт ModR/M. Т.к. один из аргументов – содержимое памяти (m8), а смещение может уместиться в 8-разрядах, то mod = 01. В поле R/M кодируем операнд – [bx+смещ8]. Его код = 7. Т.к. в шаблоне второй аргумент не регистр, то поле REG/КОП может принимать любое значение (заполним нулем). Получили ModR/M = 47. Но верными будут и значения 57, 67,77, 4F, 5F, 6F,7F.

0	1	0	0	0	1	1	1
MOD		REG/КОП			R/M		

Код команды = C6 47. Это не правильно. Построенный код не учитывает величину смещения (= 6) и значение непосредственного операнда (= 24_{16}). Исправим. Полный код команды = C6 47 06 24.

Пример 3.

Команда	Режимы адресации аргументов	Шаблон из справочника
NEG dx	neg r16	neg r/m8 F6 /3

В справочнике нет указанной команды с 16-разрядным операндом. Но команда помечена (*), т.е. 16-разрядный аргумент – допустимый операнд в этой команде. Для получения из кода в справочнике (F6) требуемого кода операции установим младший бит в 1. Получили КОП = F7. Найденный в справочнике код (F6) имеет расширение (/3). Расширение записываем в поле REG/КОП байта ModR/M. Остальные поля мы заполнять уже умеем.

1	1	0	1	1	0	1	0
MOD		REG/КОП		R/M			

Полный код команды = F7 DA.

Пример 4.

Команда	Режимы адресации аргументов	Шаблон из справочника
mov dl,24h	mov r8,im8	mov r8,im8 B0+rb

Такая запись кода (B0+rb) означает, что код команды получается сложением базового кода (B0) и кода регистра (dl=2): B0+2=B2. Байт ModR/M не нужен. Полный код команды = B2 24.

Пример 5.

Команда	Режимы адресации аргументов	Шаблон из справочника
jmp met	jmp m	jmp label E9
jmp met	jmp m	jmp short label EB

Все команды условного перехода и short переход занимают 2 байта. Из них первый байт содержит КОП, а второй – значение, равное разности между адресом метки и адресом байта следующего за командой перехода. Рассмотрим переход вперед. В этом случае метка met транслятору ещё не встречалась. Он не знает близким (near) или коротким (short) окажется этот переход, поэтому, рассчитывая на худшее будет строить близкий переход и зарезервирует под команду 3 байта. Затем вычислит адрес метки. Если

расстояние от команды до метки окажется меньше 128 байт, то КОП E9 будет заменён на EB, записан однобайтный операнд, и так как, зарезервировано было 3 байта, то третьим байтом будет КОП NOP = 90 (no operation). При переходе назад код операции сразу будет проставлен правильно, т.к. адрес метки уже известен. Алгоритм расчёта операнда не измениться, но так как адрес метки меньше, чем адрес команды+2, то разность будет отрицательной и будет записана в дополнительном коде.

Если переход NEAR, то расчёты те же, но операнд займет 2 байта.

Машинный код команды косвенного перехода будет содержать байт ModR/M.

Шестнадцатеричный редактор Hiew.

Hacker's Viewer – это шестнадцатеричный редактор, дизассемблер и ассемблер. Он позволяет просматривать файлы неограниченной длины в текстовом и шестнадцатеричном форматах, а также в режиме дизассемблера процессора 80x86. Основные возможности программы: редактирование файлов в шестнадцатеричном режиме и в режиме дизассемблера; поиск и замена в блоке; встроенный ассемблер; поиск ассемблерных команд по шаблону; поддержка различных форматов исполняемых файлов: MZ, NE, LE, LX, PE.

По умолчанию программа работает в режиме просмотра текста. Воспользуйтесь F4 для смены режима на Text, Hex, Code. Каждый из режимов поддерживает свой спектр возможностей.

ЗАДАНИЕ

1. Записать программу в машинных кодах.

– Изучить последовательность команд в мнемонических обозначениях согласно варианту (ПРИЛОЖЕНИЕ 1.1), написать, какие действия

выполняет каждая команда. Указать, какие режимы адресации используются в каждой команде.

– Построить машинный код для команд своего варианта, используя справочник (ПРИЛОЖЕНИЕ 1.2, 1.3), с объяснением хода построения (см. примеры выше).

2. Ввести программу в машинных кодах.

– Создать новый файл с расширением com (в NC / FAR / WinCmd нажать <Shift> + <F4> и ввести имя файла с расширением com).
– Вызвать Hiew.exe с параметром: Hiew.exe имя_файла.com (или в NC / FAR / WinCmd установить курсор на hiew.exe нажать <Ctrl> + <Enter>, затем на com-файл, снова <Ctrl> + <Enter> и потом <Enter>).
– Выбрать режим HEX (<F4>,<F2>).
– Перейти в режим редактирования (<F3>).
– Ввести построенный в п.1 задания машинный код.
– Сохранить результат работы (<F9>).
– Посмотреть дизассемблированные команды (<F4>, <F3>), проверить соответствие полученных команд заданным.

3. Ввести программу в мнемонических обозначениях.

– Используя Hiew.exe в режиме Decode → Asm ввести следующую программу:

1) MOV BX,110
2) MOV AX,[BX]
3) ADD AX,[BX+2]
4) MOV SI,4
5) MOV [BX+SI],AX
6) NOP
7) INT 20

– Перейти в режим HEX и ввести данные: 2301 2500 0000.
– Сохранить программу.
– Написать, что делает эта программа.
– Перечислить использованные в программе режимы адресации.

– Просмотреть машинные коды этой программы и содержимое области данных.

СОДЕРЖАНИЕ ОТЧЁТА

– Тема и цель работы; задание на лабораторную работу (свой вариант).
– Ход выполнения работы:
  – для каждой строки задания своего варианта указать команду, режимы адресации, описание действий выполняемых командой;
  – для каждой команды привести машинные коды с описанием их построения;
  – объяснить назначение двух последних команд задания и привести примеры результата выполнения этих команд для конкретных значений операндов.
– Выводы о проделанной работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как формируется физический адрес у МП i8086?
2. Как разделяются регистры МП i8086 по функциональному признаку?
3. Перечислите сегментные регистры и укажите их назначение?
4. Какие регистры могут использоваться при адресации явно (неявно)?
5. Какие режимы адресации существуют в МП i8086?
6. Какие поля может содержать машинная команда?
7. Какие режимы ввода данных доступны в Hiew?
8. Как осуществляется ввод программы в машинных кодах в Hiew?

Задания по вариантам ПРИЛОЖЕНИЕ 1.1

№ варианта	Последовательность команд	№ варианта	Последовательность команд
1	xchg ax,di not byte ptr [bx+800h] cmp bl,bh rep stosb jnz $+5 shl ax,1 xor ah, 61h	9	dec di push word ptr [bx+111h] test cl,dl std ja $-20h shr ax,cl and ch,40h
2	inc cx pop word ptr [bx+si+76h] add ah,ch cbw jbe $-5 ror al,cl or cl,5fh	10	push bx neg byte ptr [bp+di+222h] xchg al,bl lodsw jz $+60h rol dl,1 xor dl,21h
3	dec bp mul byte ptr [19h] adc dh,cl clc jmp $+161h rcr bx,1 and bl,5fh	11	pop bp not cl xor [bx+di],dx scasb jnc $-10 ror bh,cl or dl,5Ah
4	push di div byte ptr [bp+si] sub ch,dh cmc jg $+25 rcl dx,cl	12	mov di,44h neg ah and [di],cx clc jns $-16 rcl bl,1

№ варианта	Последовательность команд	№ варианта	Последовательность команд
	xor dh,35h		or cl,66h
5	pop si imul byte ptr [di+64h] sbb bl,cl cwd loop $-10 shl cx,1 or dh, 20h	13	xchg ax,cx mul cx test di,[bp+41h] stc jmp [100h] rol ch,1 and cl,7Eh

6	mov dx,20h
	idiv byte ptr es: [bp+di]
	cmp ah,dl
	cld
	jmp [bx]
	shr bx,cl
	and bl,31h
		14	inc ax
			div di
			or [bx+di+0A4Ch],bp
			lodsb
			jle $-30h
			sar ah,cl
			xor dh,27h

7	xchg ax,dx inc word ptr [si+324h] mov dh,al xlat jc $-12 sar dl,1 xor dl,55h	15	dec bx imul si xor cx,[si+63h] stc jb $-40h rcl cl,1 and bh,78h
8	inc cx dec word ptr [bp+si] test ch,dl stc jmp $+23h rcl dx,cl or cx,80h	16	pop bx inc dl mov cs:[20h],ah xlat jmp [bx+si] shl cx,1 xor dl,73h

Система команд МП i8086 ПРИЛОЖЕНИЕ 1.2

Мнемоника	КОП	*	oditszapc	Комментарии
AAA	37		?---??*?*	AL:=(AL исправленный для ASCII-сложения)
AAD	D50A		?---**?*?	AH:AL:=(AH:AL подготовить для деления BCD)
AAM	D40A		?---**?*?	AH:AL:=(AH:AL испр. для ASCII-умножения)
AAS	3F		?---??*?*	AL:=(AL исправленный для ASCII-вычитания)
ADC ac,im	14 im	*	*---*****	dst:=(src + dst + CF)
ADC r/m8,im8	80 /2 im	*		сложить, учитывая перенос, результат в dst
ADC r/m8 ,r8	10	*		
ADC r8,r/m8	12	*		
ADD ac,im	04 im	*	*---*****	dst:=(src + dst);
ADD r/m8,im8	80 /0 im	*		сложить два операнда, результат в dst
ADD r/m8 ,r8	00	*		
ADD r8,r/m8	02	*		
AND ac,im	24 im	*	0---**?*0	dst:= dst & src
AND r/m8,im8	80 /4 im	*		сброс битов dst, равных 0 в src
AND r/m8 ,r8	20	*		
AND r8,r/m8	22	*		
CALL label			---------	запомнить адрес возврата в стеке, передать управление
CALL far im	9A			вызов межсегментный прямой
CALL far r/m	FF /3			вызов межсегментный косвенный
CALL near r/m	FF/2			вызов близкий косвенный
CALL near im	E8			вызов близкий, смещение относительно следующей команды

Мнемоника	КОП	*	oditszapc	Комментарии
CBW	98		--------	Преобразовать байт в слово AH:=(заполнен битом 7 из AL)
CLC	F8		-------0	CF:=0 очистить флаг переноса
CLD	FC		-0-------	DF:=0 очистить флаг направления
CLI	FA		--0------	IF:=0 запретить маскируемые аппаратные прерывания
CMC	F5		-------*	CF:=~CF инвертировать флаг переноса
CMP ac,im	3C im	*	*---*****	сравнение выполняется как неразрушающее вычитание: флаги:=(dst - src). Влияет только на флаги.
CMP r/m8,im8	80 /7 im	*		
CMP r/m8, r8	38	*		
CMP r8,r/m8	3A	*		
CMPSB	A6		*---*****	[byte ptr DS:SI]-[byte ptr ES:DI], si,di+=Δ; Δ =±1
CMPSW	A7			[word ptr DS:SI]-[word ptr ES:DI], si,di+=Δ; Δ =±2
CWD	99		--------	word ->dword; DX:=(заполнен 15-м битом из AX)
DAA	27		?---*****	AL:=(AL исправленное для BCD-сложения)
DAS	2F		?---*****	AL:=(AL исправленное для BCD-вычитания)
DEC r/m8	FE /1	*	*---****-	dst:=(dst - 1) вычесть 1 из dst
DEC r16	48+RW			
DIV r/m8	F6 /6		?---?????	разделить аккумулятор на байтт без знака (со знаком) AL:=(AX div src8); AH:=(AX MOD src8)
IDIV r/m8	F6 /7		?---?????	

Мнемоника	КОП	*	oditszapc	Комментарии
DIV r/m16	F7 /6			AX:=(DX:AX div src16); DX:=(DX:AX MOD src16)
IDIV r/m16	F7 /7			
IMUL r8/m8	F6 /5		*---????	умножить AL на целое со знаком. Результат в AX.
IMUL r16/m16	F7 /5			DX:AX:=(AX * src16)
IN ac,im8	E4 im	*		чтение в AL/AX из порта I/O
IN ac,DX	EC im	*		AL:=[порт]; AH:=[порт+1]
INC r/m8	FE /0	*	*---***-	dst:=(dst+1)
INC r16	40+rw		*---***-	прибавить 1 к dst
INSB / INSW	6C / 6D		---------	ES:[DI]:=(байт/ слово из порта DX); DI±=Δ;
INT type			--00-----	выполнить программное прерывание: pushf; IF=0; tf=0; push CS; push IP; IP:= 0000:[type * 4]; CS := 0000:[(type * 4) + 2]
INT 3	CC		--00-----	ловушка отладчика
INTO	CE		--**-----	прерывание по переполнению (если OF=1, то INT 4)
INT im8	CD		--00-----	см. INT type, INT 3
IRET	CF		********	возврат из прерывания. (POP IP; POP CS; POPF)
Jcond s_lab	см. прило-жение № 1.3		---------	переход по условию: IP=IP+(8-битное смещение, расширенное со знаком до 16 бит)
JMP label			---------	безусловная передача управления на метку
JMP short_im8	EB			short: IP:=IP+(смещение цели, расширенное со знаком)
JMP near_im16	E9			near: IP:= near_im16

Мнемоника	КОП	*	oditszapc	Комментарии
JMP near r/m16	FF /4			near: IP:= r/m16
JMP far im16:16	EA			far: CS:=целевой_сегмент; IP:=целевое_смещение
JMP far m	FF /5			
LAHF	9F		--------	загрузить флаги в AH
LEA r16,m	8D		--------	загрузить адрес в регистр reg:=(результат вычисления исполнительного адреса)
LDS r16,m32	C5		--------	загрузить DS и reg16 из поля памяти: r:=[m16]; DS:=[m16+2]
LES r16,m32	C4		--------	загрузить ES и reg16 из поля памяти: r:=[m16]; ES:=[m16+2]
LODSB	AC		--------	загрузка байта из строки в AL(AX); AL := DS:[SI]; SI+=1;
LODSW	AD		--------	AX := DS:[SI]; SI+=2;
LOOP s_lab	E2		--------	цикл CX:=(CX-1) и short переход если CX≠0
LOOPE/LOOPZ	E1		--------	цикл CX:=(CX-1) и short переход если CX≠0 && ZF=1
LOOPNE / LOOPNZ	E0		--------	цикл CX:=(CX-1) и short переход если CX≠0 && ZF=0
MOV r/m8,r8	88	*	--------	переслать из src в dst
MOV r8,r/m8	8A	*		
MOV r/m16,seg	8C			
MOV seg,r/m16	8E			
MOV ac,m	A0	*		
MOV m,ac	A2	*		
MOV r8,im8	B0+rb			
MOV r16,im16	B8+rw			
MOV r/m8,im8	C6 im	*		

Мнемоника	КОП	*	oditszapc	Комментарии
MOVS dst,src			---------	копировать строку байт (слов) (байт: Δ=1, слово: Δ=2) es:[di]:=ds:[si]; di+=Δ; si+=Δ
MOVSB	A4			
MOVSW	A5			
MUL r/m8	F6 /4		*---????	умножить AL на значение без знака: AX:=(AL * src8)
MUL r/m16	F7 /4			умножить AX на значение без знака: DX:AX:=(AX * src16)
NEG r/m8	F6 /3	*	*---*****	dst:=(0 - dst); изменить знак
NOP	90		---------	отсутствие операции
NOT r/m8	F6 /2	*	---------	dst:=~dst (инверсия всех бит dst)
OR ac,im	24 im	*	0---**?*0	dst:=dst | src (установка битов dst, равных 1 в src)
OR r/m8,im8	80 /1 im	*		
OR r/m8 ,r8	08	*		
OR r8,r/m8	0A	*		
OUT DX,ac	EE	*	---------	вывод из AL/ AX в порт ввода-вывода
OUT im,ac	E6	*		
OUTSB	6E		---------	вывод из DS:[SI] в порт ввода-вывода
OUTSW	6F		---------	
POP m16	8F /0		---------	r/m:=SS:[SP]; SP+=2 извлечь из стека в r/m16
POP r16	58+rw			
POP ds/es/ss	1F/ 07/ 17			sreg := SS:[SP]; SP+=2; cs –недопустим
POPF	9D		*********	POP Flags: flags:=SS:[SP]; SP+=2
PUSH r16	50+rw		---------	SP-=2; SS:[SP]:=r/m/im
PUSH m16	FF /6		---------	
PUSH im16	68		---------	
PUSH CS/DS	0E/1E		---------	
PUSH SS/ES	16/06		---------	

Мнемоника	КОП	*	oditszapc	Комментарии
PUSHF	9C		--------	PUSH Flags: переслать регистр флагов в стек; SP-=2; SS:[SP]:=флаги
RCL r/m8,1	D0 /2	*	*-------*	← CF ← [7..0] ← CF циклический сдвиг влево через перенос
RCL r/m8,CL	D2 /2	*		
RCR r/m8,1	D0 /3	*	*-------*	CF → [7..0] → CF циклический сдвиг вправо через перенос
RCR r/m8,CL	D2 /3	*		
ROL r/m8,1	D0 /0	*	*-------*	← CF ← [7..0] ← CF циклический сдвиг влево
ROL r/m8,CL	D2 /0	*		
ROR r/m8,1	D0 /1	*	*-------*	CF → [7..0] → CF циклический сдвиг вправо
ROR r/m8,CL	D2 /1	*		
REP без LODS	F3		--------	(префикс) CX:=(CX-1); повторять строковую операцию пока CX ≠ 0
REP перед LODS	F2		--------	
REPE/REPNE	F3/F2		--------	(префикс) CX:=(CX-1); повторять пока (CX≠0 и ZF ? 0)
RET near im	C2		--------	выход из подпрограммы с удалением im байт из стека
RETF im	CA		--------	
RET near	C3		--------	выход из подпрограммы
RETF	CB		--------	
SAHF	9E			flags:=AH
SAL r/m8,1	D0 /4	*	*-------*	CF← [7 .. 0] ← 0 арифметический сдвиг влево
SAL r/m8,CL	D2 /4	*		
SAR r/m8,1	D0 /7	*	*-------*	[7]+[6 .. 0] → CF арифметический сдвиг вправо
SAR r/m8,CL	D2 /7	*		
SHL r/m8,1	D0 /4	*	*-------*	← CF ← [7..0] ← 0 логический сдвиг влево
SHL r/m8,CL	D2 /4	*		
SHR r/m8,1	D0 /5	*	*-------*	0 → [7..0] → CF логический сдвиг вправо
SHR r/m8,CL	D2 /5	*		

Мнемоника	КОП	*	oditszapc	Комментарии
SBB ac,im	1C im	*	*---*****	dst:=((dst - src) - CF)
SBB r/m8,im8	80 /3 im	*		вычесть, учитывая заём
SBB r/m8 ,r8	18	*		
SBB r8,r/m8	1A	*		
SCASB/SCASW	AE/AF		---------	флаги:=(рез-тат CMP DS:[DI],AL/AX); DI+=1 или 2
STC/STD	F9/FD		-*------*	установить флаг CF:=1 (переноса) / DF:=1 (направления)
STI	F1		--1------	IF:=1 разрешить маскируемые аппаратные прерывания
STOSB	AA		---------	ES:[DI]:=AL; DI+=1;
STOSW	AB		---------	ES:[DI]:=AX; DI+=2;
SUB al/ax,im	2C/2D		*---*****	dst:=(src + dst)
SUB r/m8,im8	80 /5	*		сложить два операнда, результат в dst
SUB r/m8 ,r8	28	*		
SUB r8,r/m8	2A	*		
TEST ac,im	A8 im	*	0---**?*0	неразрушающее И
TEST r/m8,im8	F6 /0 im	*		флаги:=(как для dst & src)
TEST r/m8 ,r8	84	*		
XCHG ax,r16	90+rw		---------	dst ⇔ src
XCHG r/m,r8	86	*	---------	
XLAT	D7		---------	AL:=ES:[BX+(AL)]
XOR ac,im	34 im	*	0---**?*0	dst:=(dst ^ src);
XOR r/m8,im8	80 /6 im	*		инверсия битов dst,
XOR r/m8 ,r8	30	*		равных 1 в src
XOR r8,r/m8	32	*		

Используемые обозначения:

ac	операнд в аккумуляторе – AL / AX
r	операнд в регистре – AL, AH, BL, BH, CL, CH, DL, DH, AX, BX, DX, CX, SI, DI, BP, SP
m	операнд в памяти – метка, символьное имя, переменная
seg	операнд в сегментном регистре: CS, DS, ES, SS
r/m	операнд в регистре или в памяти
im	операнд – непосредственное значение (const, имя)
scr	источник
dst	приемник
a+=b	a:=a+b
a-=b	a:=a+b
a±=b	a:=a±b
Δ	обозначено приращение на 1 или 2 (величина зависит от команды или операнда)

В четвёртом столбце показано, какое влияние команда оказывает на флаги:

?	флаг не определён;
-	флаг не изменился;
*	флаг выставлен этой командой в соответствии с результатом.

Команды, отмеченные звёздочкой в 3 столбце, могут оперировать не только 8 но и 16-разрядными операндами. Код операции для команды с 16-разрядными операндами получается из приведённого установкой младшего бита.

Команды условной передачи управления. ПРИЛОЖЕНИЕ 1.3

Команда	КОП	переход если...	Условие перехода
JA	77	выше после без знаковой арифметики	CF=0 и ZF=0
JAE	73	выше или равно для без знаковых	CF=0
JB	72	ниже (переход если Carry)	CF=1
JBE	76	ниже или равно	CF=1 или ZF=1
JC	72	Carry – перенос	CF=1
JCXZ	E3	CX=0	CX=0
JE/JZ	74	равно	ZF=1
JG	7F	больше	SF=OF & ZF=0
JGE	7D	больше или равно	SF=OF
JL	7C	меньше	ZF≠OF
JLE	7E	меньше или равно	SF≠OF | ZF=1
JNB	73	не ниже (выше или равно)	CF=0
JNBE	77	не ниже или равно (выше)	CF=0 и ZF=0
JNC	73	не устновлен флаг переноса	CF=0
JNE/JNZ	75	не равно	ZF=0
JNG	7E	не больше	SF≠OF или ZF=1
JNGE	7C	не больше или равно (меньше)	SF≠OF
JNL	7D	не меньше	SF=OF
JNLE	7F	не меньше или равно (больше)	ZF=0 и SF=OF
JNO	71	нет переполнения	OF=0
JNP	7B	нет "чётности"	PF=0
JNS	79	знаковый разряд нулевой	SF=0
JO	70	переполнение	OF=1
JP/ JPE	7A	число единичных битов чётно	PF=1
JPO/JNP	7B	сумма битов нечётна	PF=0
JS	78	знак	SF=1

Лабораторная работа № 2
ПРОЦЕСС СОЗДАНИЯ И ОТЛАДКИ ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА

Цель работы: Знакомство с методами создания и отладки программ, написанных на языке ассемблера.

ДОМАШНЯЯ ПОДГОТОВКА

- Изучить технологию разработки ассемблерных программ.
- Изучить основные форматы исполняемых файлов DOS, порядок их загрузки на исполнение.
- Подготовить ответы на контрольные вопросы.

ЗАДАЧИ

- Изучить способы ассемблирования и создания исполняемого файла с помощью программ Turbo Assembler (TASM.EXE) и Turbo Link (TLINK.EXE).
- Познакомиться с командами и интерфейсом отладчика Turbo Debbuger (TD.EXE), научиться трассировать и исправлять программы.
- Создать программу на языке ассемблера выполняющую арифметическую операцию и ввод/вывод с консоли.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Технология создания программ на языке ассемблера

Рассмотрим процесс создания программ на языке ассемблера с использованием пакета Turbo Assembler фирмы Borland International.

1. Создание исходного текста программы в операторах языка ассемблер (.asm-файл): на данном этапе можно использовать любой текстовый редактор, позволяющий создать ASCII-файл. Исходный текст

программы на языке ассемблера представляет собой обычный текстовый файл, содержащий операторы и директивы языка ассемблер.

Важно понимать что программы пакета Turbo Assembler – это DOS-приложения, поэтому к именам файлов и каталогов предъявляются требования этой ОС: в имени и расширении не должно быть русских букв и пробелов, длина имени не более 8 символов, расширения не более трёх.

2. Трансляция операторов языка в машинные коды ЭВМ. Для этого используется программа TASM, которая осуществляет трансляцию исходного текста в машинные коды и генерацию объектного модуля. В результате получается модуль (объектный файл, .obj) а, при необходимости, файл с листингом программы (.lst) и файл перекрёстных ссылок (.crf). В процессе трансляции TASM создаёт таблицу идентификаторов, которую можно представить в виде перекрёстных ссылок на метки, идентификаторы и переменные в программе. Посмотрите последние строки (Symbol Table) вашего файла листинга. Число в формате n# указывает на номер строки в листинге программы, где определён соответствующий идентификатор. Остальные числа в этой строке, указывают на номера строк, в которых используется этот идентификатор.

3. Компоновка, создание исполняемого файла (.exe или .com). Используется программа TLINK. При помощи компоновщика можно объединить несколько отдельно оттранслированных исходных модулей в один исполняемый файл.

4. При необходимости может быть выполнена трассировка полученной программы с целью поиска алгоритмических ошибок при помощи отладчика программ TD.

5. Эксплуатация. Программа загружается на исполнение с помощью стандартных средств операционной системы.

Основные форматы исполняемых файлов в MSDOS

Формат COM

Программы типа .com хранятся на диске в виде точного образа программы в памяти, поэтому .com программы ограничены единственным сегментом, то есть машинный код, данные и стек в сумме могут занимать в памяти не больше 64 Кб. Так как .com файлы не содержат никакой настроечной информации, то они компактнее эквивалентных .exe файлов, и загружаются для выполнения немного быстрее. Отметим, что DOS не пытается выяснить действительно ли файл с расширением .com содержит исполняемую программу. Программы типа .com загружаются непосредственно за префиксом сегмента программы (PSP) и, кроме того, не имеют заголовка, который может задавать другую точку входа, поэтому их начальный адрес всегда составляет 100h, что определено размером PSP = 256 (100h) байт. Максимальная длина .com- программы = 64k минус обязательное слово стека минус 256 байт PSP.

Рассмотрим подробнее, как DOS загружает .com программу:

Рис. 3. Образ памяти программы .COM

- создает PSP (256-байтную рабочую область в начале программы, содержащую важную для DOS информацию. Здесь же, со смещением 80h, находится «хвост» командной строки);
- целиком копирует .com-файл в память непосредственно за PSP;
- устанавливает все 4 сегментных регистра на начало PSP;

– устанавливает значение регистра IP в 100h, а регистр SP в FFFEh – на конец текущего сегмента.

Формат EXE

В настоящее время существуют несколько форматов исполняемых файлов с расширением EXE:

MZ – формат программы с 16-разрядным кодом для ОС MSDOS;

LX, NE, LE – форматы программ с 16-разрядным кодом для ОС Windows (в настоящее время практически не применяются);

PE – формат программ с 32-разрядным кодом для ОС Windows.

Программа для DOS формата MZ EXE содержит заголовок, занимающий не менее 512 байт: перемещаемую точку входа, начальные значения CS:IP, SS:SP, объём необходимой для загрузки программы памяти, таблицу перемещений (для настройки абсолютных ссылок после загрузки программы). Exe-программа имеет по крайней мере 2 сегмента: сегмент кода и сегмент стека.

Этапы выполняемые DOS для загрузки .exe программы:

– создаёт PSP;
– из заголовка .exe-файла определяет откуда начинается программа и загружает её после PSP;
– находит и исправляет все ссылки в программе, значение которых зависит от физического адреса, начиная с которого будет размещён сегмент;
– устанавливает значения регистров DS и ES так, чтобы они указывали на начало PSP;
– устанавливает CS на начало сегмента кода, значения IP и SS:SP устанавливаются исходя из информации заголовка, и указывают на первую исполняемую команду программы и вершину стека соответственно.

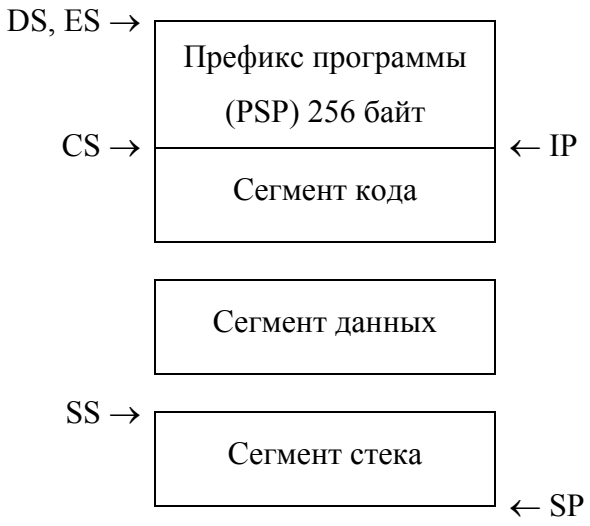


Рис. 4. Образ памяти программы .EXE

Сервисные функции MS DOS

В любой программе для ЭВМ должен быть предусмотрен ввод исходных данных и вывод результирующих. Для этого используются различные устройства ввода/вывода: клавиатура, видеокарта, диски и т.д. При создании программ на языке ассемблера МП i8086 связь с устройствами ввода/вывода осуществляется через адресное пространство ввода/вывода, а также через области памяти связанные с устройствами ввода/вывода. Так как каждое устройство имеет свой, обычно достаточно сложный, формат (протокол) передачи данных, программирование обмена данными с устройствами на языке ассемблера требует от программиста знание принципа работы устройства и протокола обмена.

Для облегчения задачи обмена данными с устройствами ввода/вывода, а также для выполнения некоторых других задач ОС MSDOS предоставляет специальные подпрограммы – сервисные функции, которые можно вызывать из программы выполняющейся под управлением ОС.

Для того чтобы программисту не требовалось знать конкретные адреса расположения сервисных подпрограмм в памяти (которые могут меняться от версии к версии), вызов сервисных подпрограмм осуществляется через программные прерывания.

Вызов программного прерывания осуществляется командой процессора INT n, где n – номер прерывания. Перед вызовом прерывания, согласно формату вызова сервисной функции, в регистры требуется занести данные необходимые для выполнения запрашиваемой операции.

Ниже приведены некоторые прерывания сервисных функций BIOS и MS DOS:

INT 10h – программы BIOS управления видеосистемой.

INT 13h – программы BIOS управления дисками.

INT 16h – программы BIOS управления клавиатурой.

INT 20h – завершение com-программы DOS.

INT 21h – диспетчер сервисных функций MS DOS (в регистре AH при вызове должен быть номер запрашиваемой функции).

Сервисные функции, необходимые для выполнения лабораторной работы, и формат их вызова приведены в приложении 2.2.

Транслятор Turbo Assembler.

Программа TASM.EXE осуществляет трансляцию программы на языке ассемблера, представленной в виде текстового файла, в машинные коды. Для того, чтобы отличать просто текстовый файл от программы на языке ассемблера, используется специальное расширение .asm (для текстовых файлов обычно используется расширение .txt).

Результатом работы программы является модуль (объектный файл, .obj) и, если необходимо, файл с листингом программы (.lst) и файл перекрёстных ссылок (.crf).

Формат командной строки транслятора TASM:

TASM [ключи_опций] имя_исходного_файла [, имя_объектного_файла [,имя_листинга [,имя_файла_перекрёстных ссылок]]]

Аргументы в квадратных скобках – необязательные параметры.

Ключи опций программы TASM приведены в ПРИЛОЖЕНИИ 2.3.

Примеры использования:

tasm.exe prog1.asm

Транслирует программу prog1.asm, результат – модуль prog1.obj.

tasm prog2, progR.obj, prog2.lst, prog2.crf

Транслирует программу prog2.asm, результат – модуль progR.obj, листинг prog2.lst, файл перекрёстных ссылок prog2.crf.

tasm.exe /zi prog3.txt, prog3.obj, prog3.lst

Транслирует программу prog3.txt, результат – модуль prog3.obj с информацией для отладчика (ключ /zi) и листинг prog3.lst.

Компоновщик Turbo Link.

Программа TLINK.EXE осуществляет связывание модулей, полученных с помощью TASM, и создание исполняемого файла.

Формат командной строки компоновщика TLINK:

TLINK [ключи] список_obj_файлов [,имя_исполняемого_файла [,имя_map_файла [,имя_lib_файла [,имя_def_файла [,имя_res_файла]]]]]

Ключи опций программы TLINK приведены в ПРИЛОЖЕНИИ 2.3.

Примеры использования:

tlink.exe prog1.obj

Результат – исполняемый файл prog1.exe.

tlink /t prog2

Результат – исполняемый файл prog2.com.

tlink.exe modul1.obj+modul2.obj, prog.exe, prog.map

Результат – исполняемый файл prog.exe, файл карты размещения prog.map.

Отладчик Turbo Debbuger.

Turbo Debugger – отладчик программ в машинных кодах из пакета TASM Borland International. Основные возможности: выполнение

трассировки программы в прямом и обратном направлении; просмотр и изменение регистров и содержимого ячеек памяти во время покомандного выполнения программы; поддержка точек останова. TD не имеет встроенного редактора текстов, и не может перекомпилировать вашу программу. Внесённые в код во время работы изменения не будут сохранены на диске. Для внесения изменений в программу в машинных кодах необходимо использовать шестнадцатеричный редактор (например Hiew).

Формат командной строки отладчика TD:

TD [имя_исполняемого_файла]

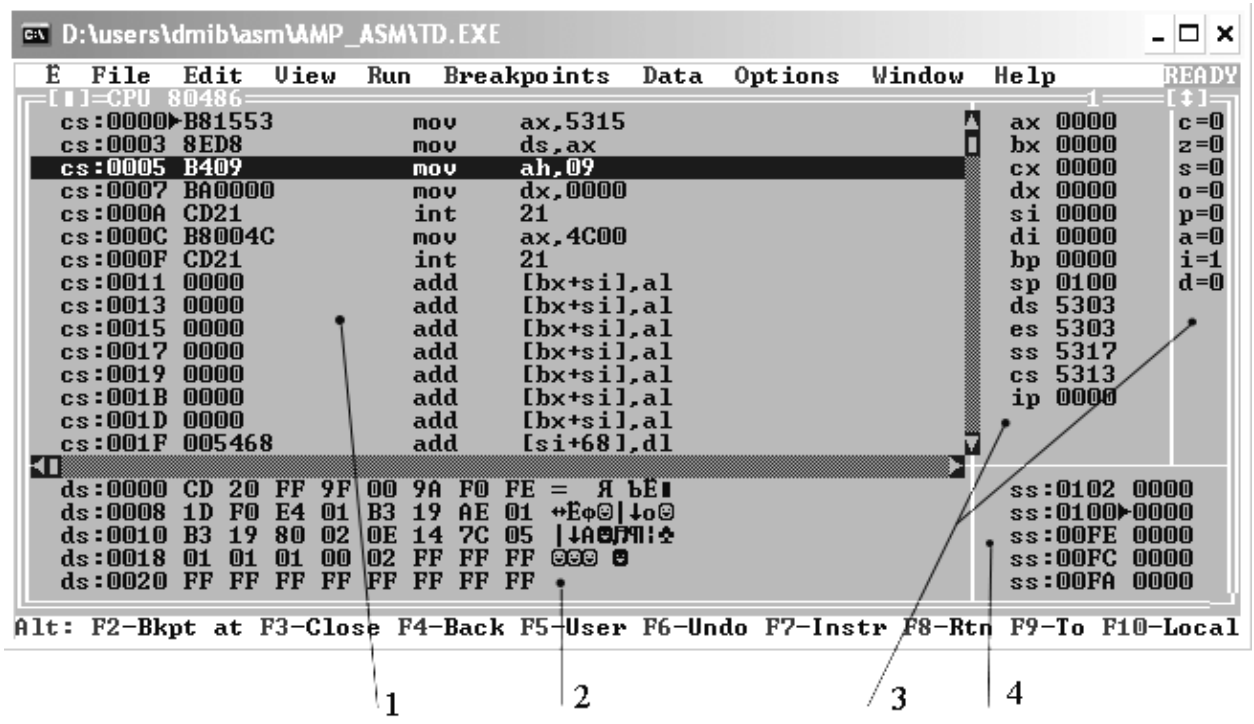
где необязательный параметр имя_исполняемого_файла – исполняемый файл, который загружается в отладчик для трассировки.

Отладчик TD имеет текстовый псевдографический оконный интерфейс.

Основные элементы интерфейса:

Окно дизассемблера (1) – предназначено для отображения программы в памяти: для каждой команды указан адрес памяти, машинный код и мнемоническое обозначение. Специальный указатель ► указывает на текущую исполняемую команду. В начале указатель установлен на первую исполняемую команду программы.

Окно шестнадцатеричного просмотра (дампа) (2) – предназначено для отображения участка памяти в шестнадцатеричном виде, обычно его настраивают на сегмент данных программы.



Окно регистров и окно регистра флагов (3) – предназначено для просмотра регистров процессора в ходе выполнения программы и, при необходимости, внесения изменений в хранимые значения;

Окно стека (4) – предназначено для отображения стека программы: указывается адрес и значение, хранимое в стеке. Специальный указатель ► указывает на вершину стека.

Основные клавиши управления:

<TAB>, <Shift+TAB> – переключение между окнами отладчика;

<F4> – выполнение программы до курсора (выделенной строки) в окне дизассемблера;

<F5>, <Ctrl>+<F5> – изменение размеров окна

<F7> – выполнение одной инструкции с входом в подпрограммы;

<F8> – выполнение одной инструкции без входа в подпрограммы;

<F9> – запуск программы на исполнение;

<F10> – выход в меню;

<Ctrl>+<F2> – сброс режима трассировки (перезагрузка программы);

<Alt>+<F5> – переключение между отладчиком и окном отлаживаемой программы (пользовательский экран);

<Ctrl>+<G> – переход на заданный адрес памяти в окнах дизассемблера, дампа и стека.

<Ctrl>+<Z> – выход из программы TD;

ЗАДАНИЕ

1. Создать программу формата .exe на языке ассемблера.

В текстовом редакторе наберите приведённый в примере текст программы на языке ассемблера. Сохраните его в файл first.asm;

Посмотрите как набран текст примера: отступы в начале строки облегчают чтение текста, поиск меток. Между мнемоникой и операндами и перед комментариями тоже обычно делают отступ.

Обратите внимание на имена сегментов: они произвольные и смысл в названии – только для пользователя. Назначение сегмента указано в директиве ASSUME.

```
begin segment                   ; Сегмент кода программы
assume cs: begin, ds:dates, ss: komod
start:
        mov   ax, dates         ; Настройка DS на начало сегмента данных
        mov   ds, ax
        mov   ah, 9             ; Функция 9 сервиса DOS:
        mov   dx, offset Msg    ; вывод строки с указанного
        int   21h               ; в ds:dx адреса до символа '$'
        mov   ax, 4c00h         ; Завершение программы с кодом 0
        int   21h
begin ends
dates  segment                  ; Сегмент данных
Msg  db  'Hello! ', 13,10,'$'
```

```
dates  ends
komod       segment      stack  ; Сегмент стека
dw  128  dup (?)                ; под стек отводится 128 слов
komod       ends
        end    Start
```

Оттранслируйте исходный текст и скомпонуйте из него исполняемый .exe файл:

```
TASM       first,,,
TLINK      first,,
```

Выполните полученный EXE - файл.

2. Создать программу формата .com на языке ассемблера.

Чтобы получить программу в формате .com необходимо внести в текст программы небольшие изменения (сгруппировать все сегменты в один):

Пример .com программы: Способ 1

```
begin segment
        org 100h
assume cs:begin, ds:begin
start:  mov   dx, offset Msg
        mov   ah,9
        int     21h
        int     20h


Msg  db      'Hello!',13,10,'$'
Begin ends
        end start
```

Пример .com программы: Способ 2

```
MyGrp group begin, dates
begin segment
        org 100h
assume cs: MyGrp, ds: MyGrp
start:  mov   dx, offset mygrp: Msg
        mov   ah,9
        int     21h
        int     20h
begin  ends
dates  segment
Msg    db     'Hello!', 13,10,'$'
Dates  ends
        end start
```

Внесите указанные изменения и сформируйте .com – файлы.

3. Изучить режимы выполнения программы в TD.

В текстовом редакторе наберите следующий исходный текст программы на языке ассемблера. Сформируйте .exe – файл.

```
code    segment
assume cs: code, ds:data, ss: stek
start:  mov     ax, data
        mov     ds, ax
        mov     ax,word ptr [X]         ; первая часть
        mov     bx,[Y]
        add     ax,bx
        mov     [Result],ax
        mov     ax,[Y+2]
        adc     ax,word ptr [X+2]
        mov     [Result+2],ax
        mov     ah,1                    ; вторая часть
        int     21h
        mov     [Chr],al
        mov     ah, 9
        mov     dx, offset Msg
        int     21h
        mov     ax, 4c00h
        int     21h
code    ends
data    Segment
X       dd      123456h         ; число 123456h
Y       dw      0FF88h,0077h    ; число 77FF88h
Result  dw      0,0             ; результат
Msg     db      13,10,'Enter:'
```

```
Chr     db      '0', '$'
data    ends
stek    segment stack
        dw      128 dup (?)
stek    ends
end     Start
```

Запустите TD с именем полученной программы в командной строке.

Выполните трассировку программы под управлением отладчика, отслеживая изменения значений регистров и данных, выясните, что делает программа в первой и во второй части;

4. Согласно варианту (ПРИЛОЖЕНИЕ 2.1) написать программу, по примеру в пункте 3:

- выполняющую заданную арифметическую операцию;
- выполняющую заданную операцию ввода/вывода.

СОДЕРЖАНИЕ ОТЧЁТА

- Тема и цель работы, задание на лабораторную работу (свой вариант)
- Ход выполнения работы, включая:
  - Описание процесса создания программ формата .EXE и .COM при помощи языка ассемблера;
  - Описание процесса трассировки программы-примера (пункт 3 задания), изменения значений регистров и данных в процессе трассировки программы.
- Листинг (lst-файл) разработанной программы (пункт 4 задания) с комментариями на каждой строке.
- Выводы о проделанной работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Из каких шагов состоит процесс создания программ на языке ассемблера?
2. Чем отличаются .com и .exe программы (структура, размер)?
3. Что собой представляет объектный файл (модуль)?
4. Для чего предназначена программа TASM?
5. Какая программа создает исполняемый файл?
6. Что такое сервисные функции и как они вызываются?
7. Какие действия можно выполнить при помощи программы TD?
8. Какие главные элементы интерфейса программы TD?
9. Какие функции MSDOS осуществляют вывод на экран?
10. Какие функции MSDOS осуществляют ввод с клавиатуры?
11. Сколько байт памяти занимают переменные X и Y из п.3 задания? Как (какими командами) осуществляется обращение к памяти поименованной X и Y в программе? Чем отличаются методы доступа к этой памяти? Какой из методов вам понятнее? Приведите примеры ситуаций (задач), когда может понадобиться первый и второй метод обращений к данным в памяти.

Задания по вариантам ПРИЛОЖЕНИЕ 2.1

№ вари-анта	Арифметическая операция	Операция ввода-вывода	№ вари-анта	Арифметическая операция	Операция ввода-вывода
1.	Res=X+Y x,y – трехбайтные	Ввод символа	9.	Res=X-Y x,y – трехбайтные	Вывод строки
2.	Res=X-Y x,y –четырехбайтные	Вывод символа	10.	Res=X+Y+Z x,y,z – четырехбайтные	Ввод символа
3.	Res=X+Y+Z x,y – однобайтные, z – двухбайтное	Вывод строки	11.	Res=X+Y+Z x – однобайтное, y,z – двухбайтные	Вывод символа
4.	Res=X+Y+Z x,y – двухбайтные, z – четырехбайтное	Ввод символа	12.	Res=X+Y+Z x – двухбайтное, y,z – четырехбайтные	Вывод строки
5.	Res=X+Y+Z x,y – однобайтные, z – четырехбайтное	Вывод символа	13.	Res=X+Y+Z x – однобайтное, y,z – четырехбайтные	Ввод символа
6.	Res=X-Y+Z x,y – однобайтные, z – двухбайтное	Вывод строки	14.	Res=X-Y+Z x – однобайтное, y,z – двухбайтные	Вывод символа
7.	Res=X-Y+Z x,y – двухбайтные, z – четырехбайтное	Ввод символа	15.	Res=X-Y+Z x – двухбайтное, y,z – четырехбайтные	Вывод строки
8.	Res=X-Y+Z x,y – однобайтные, z – четырехбайтное	Вывод символа	16.	Res=X-Y+Z x – однобайтное, y,z – четырехбайтные	Ввод символа

Некоторые сервисные функции MS DOS ПРИЛОЖЕНИЕ 2.2

Функция вызывается через 21h прерывание. Номер функции должен при вызове находиться в AH.

AH	Функция	Вход	Выход
1	Ввод символа с эхом с клавиатуры	-----	AL=символ
2	Вывод символа на экран	DL=символ	-----
6	Ввод/вывод символа на CON без контроля	DL=символ	при AL=FFh ввод: -> AL = символ + ZF=0 =успех, иначе вывод. Если символа нет функция завершается
7	Ввод символа с CON без эха и контроля	-----	AL=символ
8	Ввод символа без эха с CON	-----	AL=символ
9	Вывод строки на CON (до '$')	DS:DX = адрес строки	-----
A	Чтение строки в буфер	DS:DX=адрес буфера	заполненный буфер

Формат	max	кол	s1,s2,s3,...	0D	.....
буфера:			Прочитанные символы	CR	не определено

MAX - число символов, на которое рассчитан буфер;

КОЛ - количество введенных символов без учёта CR;

s1,s2,s3,... - прочитанные символы, включая завершающий CR (признак конца введенной строки).

4C	Завершить процесс с кодом возврата	AL=код возврата	-----

Ключи опций программ TASM и TLINK ПРИЛОЖЕНИЕ 2.3

Ключи TASM

/a	Сегменты в объектном файле размещать в алфавитном порядке;
/s	Сегменты в объектном файле следуют в порядке их описания в программе;
/c	Указание на включение в файл листинга информации о перекрёстных ссылках;
/dимя[= знач.]	Определяет идентификатор. Это эквивалент директивы ассемблера =, как если бы она была записана в начале исходного текста программы;
/e	Генерация инструкций эмуляции операций с плавающей точкой;
/r	Разрешение трансляции действительных инструкций с плавающей точкой, которые должны выполняться реальным арифметическим сопроцессором;
/h, /?	Вывод на экран справочной информации;
/iпуть	Задаёт путь к включаемому по директиве INCLUDE файлу. Синтаксис аргумента “путь” такой же, как для команды PATH файла autoexec.bat;
/jдирек-тива	Определяет директивы, которые будут транслироваться перед началом трансляции исходного файла программы на ассемблере. В директиве не должно быть аргументов;
/khn	Задаёт максимальное количество идентификаторов, которое может содержать исходная программа, то есть задаёт размер таблицы символов транслятора. По умолчанию может быть до 16384 идентификаторов. Это значение можно увеличить до 32 768 или уменьшить до n. Сигналом к тому, что необходимо использовать данный параметр, служит появление сообщения “Out of hash space” (“Буферное пространство исчерпано”);
/l, /la	/l - создать файла листинга, даже если он не “заказывается” в командной строке; /la - показать в листинге код, вставляемый транслятором для организации интерфейса с языком высокого уровня по директиве MODEL;
/ml	Различать во всех идентификаторах прописные и строчные буквы;

/mx	Различать строчные и прописные символы во внешних и общих идентификаторах. Это важно при компоновке с программами на тех языках высокого уровня, в которых строчные и прописные символы в идентификаторах различаются;
/mu	Воспринимать все символы идентификаторов как прописные;
/mvn	Определение максимальной длины идентификаторов. Минимальное значение n = 12;
/mn	Установка количества (n) проходов транслятора TASM. По умолчанию транслятор выполняет один проход. Максимально при необходимости можно задать выполнение до 5 проходов;
/n	Не выдавать в файле листинга таблицы;
/os, /o,	/op, /oi - генерация оверлейного кода;
/p	Проверять наличие кода с побочными эффектами при работе в защищенном режиме;
/q	Удаление из объектной программы лишней информации, ненужной на этапе компоновки;
/t	Подавление вывода всех сообщений при условном ассемблировании, кроме сообщений об ошибках;
/w0, /w1, /w2	Генерация предупреждающих сообщений разного уровня полноты: w0 – сообщения не генерируются; w1, w2 – сообщения генерируются;
/w-xxx, /w+xxx	Генерация предупреждающих сообщений класса xxx (эти же функции выполняют директивы WARN и NOWARN). Знак “-” означает “запрещена”, а “+” - “разрешена”.
/x	Включить в листинг все блоки условного ассемблирования для директив IF, IFNDEF, IFDEF и т. п., в том числе и не выполняющиеся;
/z	При возникновении ошибок наряду с сообщением о них выводить соответствующие строки текста;
/zi	Включить в объектный файл информацию для отладки;
/zd	Поместить в объектный файл информацию о номерах строк, что

необходимо для работы отладчика на уровне исходного текста программы;

/zn Запретить помещение в объектный файл отладочной информации.

Ключи TLINK (чувствительны к регистру!)

/x Не создавать файл с картой памяти (map)

/m Создать файл карты

/s Создать файл карты памяти с дополнительной информацией о сегментах (адрес, длина в байтах, класс, имя сегмента и т. д.)

/l Создать раздел в файле карты с номерами строк

/L Спецификация пути поиска библиотеки

/n Не использовать библиотеки

/c Различать строчные и прописные буквы в идентификаторах

/v Включить отладочную информацию в исполняемый файл

/3 Поддержка 32-битного кода

/d Предупреждать о дублировании идентификаторов в подключаемых библиотеках

/t Создать файл типа .com (по умолчанию .exe)

/i Инициализировать все сегменты

/o Ключ для оверлейных перекрытий

/Txx Выбрать тип выходного файла: /Tdx - DOS (по умолчанию); /Twx – Windows; (третья буква может быть c=COM, e=EXE, d=DLL)

/P [=NNNNN] Упаковка кодового сегмента

/A= NNNN Установить выравнивание адресов в новом EXE сегменте

Лабораторная работа № 3

ОРГАНИЗАЦИЯ ПОДПРОГРАММ НА ЯЗЫКЕ АССЕМБЛЕРА

Цель работы: Изучение модульного программирования на языке ассемблера.

ДОМАШНЯЯ ПОДГОТОВКА

- Изучить способы организации подпрограмм на языке ассемблера.
- Изучить способы передачи аргументов подпрограмме.
- Изучить безусловные и условные команды передачи управления.
- Подготовить ответы на контрольные вопросы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Организация ассемблерных подпрограмм.

Декомпозиция задачи на отдельные программные модули упрощает её отладку, тестирование и модификацию.

Подпрограмма (процедура или функция) представляет собой группу команд для решения некоторой подзадачи и обладает средствами получения управления из точки вызова задачи более высокого уровня и возврата управления в эту точку. Группу команд, образующих подпрограмму, можно никак не выделять в тексте программы. Однако в ассемблере принято оформлять подпрограммы специальным образом:

```
имя_подпрограмы PROC [модификатор]
  тело_подпрограммы
имя_подпрограмы ENDP
```

Директивы PROC и ENDP определяют соответственно начало и конец подпрограммы и должны начинаться одним и тем же именем. Необязательный параметр модификатор может принимать значения FAR или NEAR и характеризует возможность обращения к подпрограмме из другого сегмента кода. Этот параметр определяет тип всех команд RET

в блоке. По умолчанию модификатор принимает значение NEAR. Переход к подпрограмме называется вызовом и выполняется командой CALL. Команда вызова подпрограммы CALL передаёт управление с автоматическим сохранением в стеке адреса возврата (адреса команды, находящейся после команды CALL). При внутрисегментном вызове подпрограммы (NEAR) в стеке сохраняется как адрес возврата только содержимое IP. В случае межсегментного вызова (FAR) в стеке необходимо сохранить содержимое регистровой пары CS:IP. Имя подпрограммы считается меткой и её можно указывать в командах перехода.

Подпрограмма может размещаться в любом месте сегмента кода программы, но так, чтобы на неё случайным образом не попало управление или в другом модуле (файле). Для доступа к подпрограмме из другого модуля её имя должно быть «видимо». Для этого в модуле с подпрограммой имя процедуры должно быть объявлено глобальным:

PUBLIC имя [,имя...].

В модуле с основной программой нужно оповещение об использовании внешнего имени и информация о дальности перехода:

EXTRN имя:тип [,имя:тип...].

Аргументы подпрограммы – это данные, которые требуются для выполнения возложенных на модуль функций и расположенные вне этого модуля. Аргументы подпрограмме можно передать по ссылке или по значению; в регистрах, стеке, глобальных переменных.

При передаче аргументов по ссылке подпрограмме сообщают адрес, по которому можно взять данные. При передаче по значению подпрограмма, в заранее оговоренном месте, получает копию аргумента; оригинал остаётся недоступным и не изменяется.

Старайтесь не передавать аргументы подпрограмме через глобальные переменные. Это делает подпрограмму нереентерабельной.

Если подпрограмма получает небольшое число аргументов, то регистры – идеальное место для их передачи. В регистрах можно вернуть и результаты работы подпрограммы. Примерами служат практически все вызовы прерываний MS DOS и BIOS.

Другой способ – передавать параметры через стек. В этом случае основная программа записывает фактические параметры в стек и вызывает подпрограмму; подпрограмма работает с параметрами и, возвращая управление, очищает стек. Этот способ используется трансляторами с языков высокого уровня. Для чтения параметров из стека обычно используют регистр BP, в который помещают адрес вершины стека после входа в подпрограмму. Удалять параметры из стека должна или подпрограмма командой “RET число_байт” или вызывающая программа. Первый способ короче, более строгий и используется при реализации подпрограмм в языке Pascal. Но если за освобождение стека отвечает вызывающая программа, то становится возможным последовательными командами CALL вызывать несколько подпрограмм с одними и теми же параметрами. Этот способ дает больше возможностей для оптимизации и используется в языке C.

Пример подпрограммы вычисления n! (факториала числа n).

```
public   Fact                ; Дано в BX n
code     segment             ; Результат в AX
assume   cs:code             ; Портит DX
FACT     proc     Near
         CMP      BX,1
         JA       REPEAT
         MOV      AX,1
         RET
REPEAT:
         PUSH     BX
         DEC      BX
```

```
        CALL    FACT
        POP     BX
        MUL     BX
        RET
        endp    FACT
code    ends
        End
```

Фрагмент программы, вызывающей подпрограмму Fact

```
EXTRN   Fact:Near
Code    segment
assume  cs:code, ss:stek
start:  ...
        MOV     BX,4
        CALL    FACT            ; Вычислить 4!
        ...
Code    ends
        ...
        End     Start
```

ЗАДАНИЕ

1. Разработать **подпрограмму** проверки условия (по варианту задания), аргументы подпрограмме передать заданным (ПРИЛОЖЕНИЕ 3.1) способом.
2. Разработать **подпрограммы** ввода и вывода 16-ти разрядного числа в десятичной системе счисления (диапазон 0..65535) по примеру (ПРИЛОЖЕНИЕ 3.2).
3. Разработать COM или EXE-программу, выполняющую ввод чисел, проверку условия и вывод результата, использующую созданные подпрограммы. Подпрограммы и основная программа должны находиться в разных файлах.

СОДЕРЖАНИЕ ОТЧЁТА

Тема и цель работы; задание на лабораторную работу (свой вариант). Текст разработанной программы на языке ассемблера (основной программы и модулей) с комментариями на каждой строке. Выводы из проделанной работы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как оформляется подпрограмма на языке ассемблера?
2. На выполнение каких команд влияют параметры FAR и NEAR?
3. Для чего используются директивы EXTRN и PUBLIC?
4. Как осуществляется вызов и возврат из подпрограммы?
5. Какие существуют способы передачи аргументов подпрограмме?
6. Можно ли перейти на подпрограмму, используя команду JMP?
7. Как осуществить сборку многомодульной программы?
8. Какие команды могут использоваться для проверки условий если регистры AX и BX содержат знаковые данные, а CX и DX - беззнаковые:
   а) значение в DX больше, чем в CX?
   б) значение в BX меньше, чем в AX?
   в) CX содержит нуль?
   г) значение в BX равно или больше, чем в AX?
   д) значение в DX равно или меньше, чем в CX?
9. Какие флаги воздействуют следующие события, какое значение этих флагов, и какие команды выполняют переход по данному событию: а) произошло переполнение; б) результат отрицательный; в) результат нулевой.
10. Почему для сравнения чисел со знаком и чисел без знака применяют разные команды?

Задания по вариантам ПРИЛОЖЕНИЕ 3.1

№ варианта	Операция	Передача параметров	№ варианта	Операция	Передача параметров
1	Ввод: x, y Вывод: (x>y) = (Да/Нет)	Через регистры по ссылке	9	Ввод: x Вывод: x<1000 = (Да/Нет)	Через регистры по ссылке
2	Ввод: x Вывод: (x=0) = (Да/Нет)	Через стек по ссылке	10	Ввод: x Вывод: x>1000 = (Да/Нет)	Через стек по ссылке
3	Ввод: x, y Вывод: Max(x,y)	Через регистры по значению	11	Ввод: x Вывод: x<-1000 = (Да/Нет)	Через регистры по значению
4	Ввод: x, y Вывод: Min(x,y)	Через стек по значению	12	Ввод: x Вывод: x>-1000 = (Да/Нет)	Через стек по значению
5	Ввод: x Вывод: (x<0) = (Да/Нет)	Через регистры по ссылке	13	Ввод: x, y Вывод: x+y>80 = (Да/Нет)	Через регистры по ссылке
6	Ввод: x Вывод: (x>0) = (Да/Нет)	Через стек по ссылке	14	Ввод: x, y Вывод: x+y=100 = (Да/Нет)	Через стек по ссылке
7	Ввод: x, y Вывод: (x and y) = 0 = (Да/Нет)	Через регистры по значению	15	Ввод: x, y Вывод: x-y=0 = (Да/Нет)	Через регистры по значению
8	Ввод: x, y Вывод: (x or y) =0 = (Да/Нет)	Через стек по значению	16	Ввод: X, Y Вывод: (x xor y) =0 = (Да/Нет)	Через стек по значению

Модуль ввода/вывода десятичных чисел ПРИЛОЖЕНИЕ 3.2

```
public InputInt, OutputInt
code segment public
assume cs:code, ds:data
InputInt proc near          ; Подпрограмма ввода числа
        push bx             ; в десятичной системе счисления
        push cx             ; в диапазоне 0-65535 с консоли(клавиатуры).
        push dx             ; На выходе в регистре ax
        push si             ; находится введенное число.
        mov dx,offset strdsc
        mov ah,0Ah
        int 21h
        mov dl,0Ah
        mov ah,2
        int 21h

        xor ax,ax
        xor cx,cx
        mov cl,[strdsc+1]
        mov si,offset strbuf
        mov bx,10
s1:
        mul bx
        mov dl,[si]
        inc si
        sub dl,30h
        add ax,dx
        loop s1
        pop si
        pop dx
        pop cx
        pop bx
        ret
InputInt  endp
```

```
OutputInt proc near          ; Подпрограмма вывода десятичного числа
        push ax              ; в десятичной системе счисления
        push bx              ; в диапазоне 0-65535 на экран.
        push cx              ; На входе в регистре ax должно
        push dx              ; находится выводимое число.
        mov bx,10
        xor cx,cx

n1:     xor dx,dx
        div bx
        inc cx
        push dx
        or ax,ax
        jnz n1

        mov bx,offset strbuf
n2:     pop dx
        add dl,30h
        mov [bx],dl
        inc bx
        loop n2
        mov byte ptr [bx],'$'

        mov dx,offset strbuf
        mov ah,9
        int 21h
        pop dx
        pop cx
        pop bx
        pop ax
        ret
        endp OutputInt
code ends
```

```
data segment public
strdsc db 6,0
strbuf db 6 dup (?)
data ends
end
```

Пример программы, использующей модуль:

```
extrn InputInt:near, OutputInt:near

code segment public
assume cs:code, ds:data, ss: stek
start:
        mov ax,data
        mov ds,ax
        call InputInt           ; ввод числа
        call OutputInt          ; вывод числа
        mov dx,offset strend    ; перевод курсора на следующую строку
        mov ah,9                ;
        int 21h                 ;

        mov ax,4c00h
        int 21h
code ends

data segment public
strend  db 13,10,'$'
data ends

Stek segment stack
        dw      128 dup (?)
Stek ends
end start
```

Список рекомендуемой литературы

1. Юров, В. И. Assembler: учебник для вузов. 2-е изд.– СПб: Питер, 2003. – 637 с.
2. Зубков, С.В. Assembler для DOS, Windows и Unix. – М.: ДМК Пресс, 2000. – 608 с.
3. Финогенов, К.К. Использование языка ассемблера: уч. пособие. – М.: Горячая линия–Телеком, 2004. – 438 с.
4. Пильщиков, В.Н. Программирование на языке Ассемблера IBM PC. – М.: Диалог МИФИ, 1996. – 288 с.
5. Григорьев, В.Л. Микропроцессор i486. Архитектура и программирование (в 4-х книгах). – М.: ГРАНАЛ, 1993.– 382 с.
6. Майко, Г.В. Ассемблер для IBM PC.– М.:Бизнес-Информ; Сирин, 1997. – 212 с.
7. Скэнлон, Л. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера: пер. с англ. – М.: Радио и связь, 1991. – 336 с.
8. Абель, П. Язык ассемблера для IBM PC и программирование: пер. с англ. – М: Высшая школа, 1992 – 447 с.
9. Брэдли, Д. Программирование на языке ассемблера для персональной ЭВМ фирмы IBM: пер. с англ. – М.: Радио и связь, 1988. – 448 с.
10. Сван, Т. Освоение Turbo Assembler. – Киев: Диалектика, 1996. – 544с.

Оглавление