МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ имени академика С.П.Королева»
(Самарский университет)

К.Е.Климентьев

АЗЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ АССЕМБЛЕРА ДЛЯ ПРОЦЕССОРОВ СЕМЕЙСТВА INTEL

Рекомендовано редакционно-издательской комиссией Института информатики, математики и электроники Самарского университета в качестве методических указаний к лабораторному практикуму по курсам «Операционные системы» и «Безопасность операционных систем», которые изучаются в рамках специальностей «Комплексное обеспечение информационной безопасности» и «Информационная безопасность», а так же иным курсам аналогичной тематики для тематически близких специальностей.

Самара 2018

Оглавление

Задание на лабораторную работу……..
Введение……..
1. Общая характеристика программно-аппаратной среды…….
1.1. Общая модель вычислительной системы……..
1.2. Системная память……
1.3. Модель процессора…….
1.4. Регистр флагов…….
1.5. Режимы работы процессора……
1.6. Прерывания и исключения…….
2. Некоторые ассемблерные команды……..
2.1. Команды пересылки данных…….
2.2. Арифметические и логические команды……..
2.3. Команды перехода……
2.4. Прочие команды……
3. Способы адресации операндов……..
4. Отличия синтаксиса AT&T от Intel……
5. Некоторые приемы программирования на языке ассемблера…….
5.1. Как организовать цикл…….
5.2. Как просканировать массив чисел или строку……..
5.3. Как сложить или вычесть два числа……...
5.4. Как разделить или умножить число на степень двойки……
5.5. Как поменять местами два операнда……..
5.6. Как проверить конкретный бит операнда……..
5.7. Как установить в 1 или сбросить в 0 конкретный бит операнда….
5.8. Как обратиться к операционной системе……..
6. Примеры программ ……
6.1. Сортировка массива 4-байтовых чисел……..
6.2. Вывод на экран через обращение к ОС……..
Литература……
Приложение А. Варианты индивидуальных заданий……..
Приложение Б. Справочное ……..

Задание на лабораторную работу

Ц_е_л_ь л_а_б_о_р_а_т_о_р_н_о_й р_а_б_о_т_ы – ознакомиться с азами программирования на языке ассемблера для INTEL-совместимых процессоров.

З_а_д_а_н_и_е – написать и продемонстрировать преподавателю программу на языке ассемблера в соответствии с индивидуальными заданиями, приведенными в Приложении.

При этом:

- работу оформить как часть программы на языке высокого уровня (например, C/C++, Delphi и т.п.);

- вывод результатов на экран выполнить путем низкоуровневого обращения к операционной системе;
- использовать операционные системы Windows или Linux.

Введение

Ассемблер (от англ. to assemble – собирать вместе, монтировать) – системная программа для ЭВМ, решающая три задачи: 1) компиляцию фрагментов программы, написанных на специализированном языке программирования, в машинный код; 2) компоновку целой программы из отдельных фрагментов; 3) загрузку машинного кода программы в память ЭВМ.

В настоящее время программа-ассемблер вышла из употребления, задачи (1) и (2) решаются «компилятором» и «компоновщиком», а задача (3) – специальным компонентом операционной системы – «загрузчиком».

Машинный код программы – представление программы в виде набора числовых команд, «понятных» центральному процессору ЭВМ.

Язык ассемблера – язык программирования, «понятный» человеку, команды которого соответствуют числовым командам машинного кода.

1. Общая характеристика программно-аппаратной среды

1.1. Общая модель вычислительной системы

Упрощенная модель вычислительной системы изображена на рис. 1.

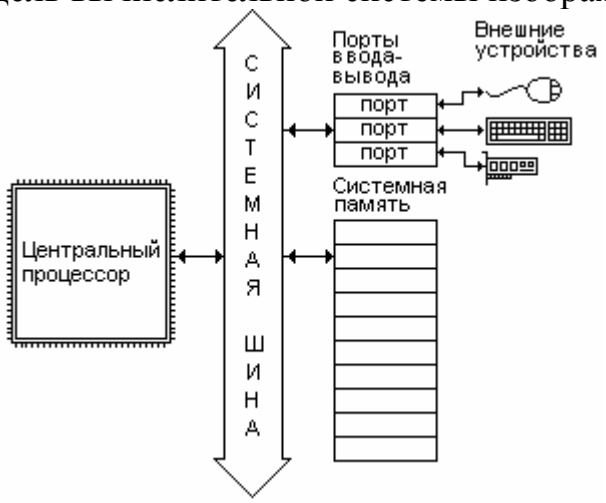

Рис. 1. Архитектура вычислительной системы

1.2. Системная память

Системная память – множество одинаковых ячеек вместимостью один *байт*. В каждой ячейке хранится число, представленное в виде группы из 8 битов.

Адрес ячейки – ее номер при последовательной нумерации от 0 до максимума. Полный набор адресов и соответствующих им ячеек памяти образует *адресное пространство* ЭВМ,

Наборы чисел, хранящихся в ячейках памяти, представляют собой либо *программы*, выполняемые процессором, либо *данные*, обрабатываемые процессором. Программы и данные друг от друга аппаратно неразличимы.

Элементы системной памяти, которые могут быть считаны или записаны одной машинной командой:

- байт (8 бит) – минимальный элемент;
- слово (16 бит);
- двойное слово (32 бита);
- четверное слово (64 бита).

Порты ввода-вывода – ячейки памяти, которые отображаются на внутренние регистры контроллеров внешних устройств.

1.3. Модель процессора

Процессор может быть представлен, как совокупность (см. рис. 2):

- арифметико-логического устройства (АЛУ);
- внутренних регистров.

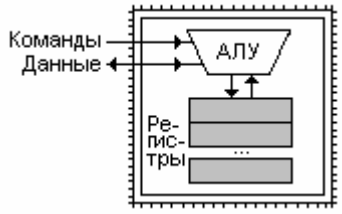

Рис. 2. Упрощенная модель процессора

Арифметико-логическое устройство (*АЛУ*) – подсистема процессора, выполняющая команды.

Регистры процессора – ячейки памяти, расположенные внутри процессора. Каждый регистр способен хранить число, представленное в виде группы битов. Условно регистры могут быть разделены на группы:

- регистры общего назначения (РОН) – используются при обработке данных;
- сегментные регистры – хранят или указывают на старшую часть адреса системной памяти;
- служебные и специальные регистры.

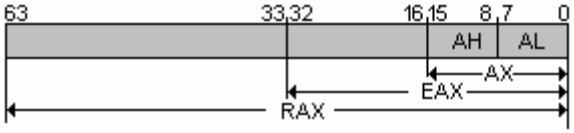

Рис. 3. Пример «составного» регистра

Часть регистров являются составными частями других регистров (см. рис. 3 и табл. 1).

Таблица 1 – некоторые регистры

64 бита	**32 бита**	**16 битов**	**Биты 8..15**	**Биты 0..7**	**Описание**
RAX	EAX	AX	AH	AL	Регистр общего назначения
RBX	EBX	BX	BH	BL	Регистр общего назначения
RCX	ECX	CX	CH	CL	Регистр общего назначения
RDX	EDX	DX	DH	DL	Регистр общего назначения
RSI	ESI	SI	нет	нет	Регистр общего назначения
RDI	EDI	DI	нет	нет	Регистр общего назначения
RBP	EBP	BP	нет	нет	Регистр общего назначения
RSP	ESP	SP	нет	нет	Указатель стека
RIP	EIP	IP	нет	нет	Указатель текущей команды
нет	нет	DS	нет	нет	Сегментный регистр
нет	нет	ES	нет	нет	Сегментный регистр
нет	нет	FS	нет	нет	Сегментный регистр

нет	нет	SS	нет	нет	Сегментный регистр
нет	нет	CS	нет	нет	Сегментный регистр

1.4. Регистр флагов

Регистр флагов представляет собой «регистр управления и состояния» (РУС). Установка в 1 или сброс в 0 некоторых битов (управляющих флагов) включает или выключает различные режимы работы процессора. Другие биты (флаги состояния) характеризуют «мгновенное» состояние процессора. Они автоматически изменяются после выполнения арифметических или логических команд. Примеры:

- бит 0 (C, CF или CARRY) – флаг переноса, содержит бит, вытесняемый из старшего или младшего бита результата во время арифметических или логических операции и во время операций сдвига, так же часто служит для индикации программных ошибок;
- бит 2 (PF) – бит четности, устанавливается при четном количестве единичных битов в числовом результате выполнения операции арифметической или логической;
- бит 4 (AF) – вспомогательный перенос;
- бит 6 (ZF) – бит нуля, устанавливается, когда результат операции равен 0;
- бит 7 (SF) – бит знака, устанавливается, когда старший бит результата равен 1;
- бит 8 (TF) – бит трассировки, включает/выключает аппаратную трассировку программ;
- бит 9 (IF) – включает/выключает реагирование процессора на сигналы прерывания;
- бит 10 (DF) – бит направления, позволяет включать увеличение или уменьшение регистра-счетчика итераций цикла;
- бит 11 (OF) – бит переполнения, устанавливается в 1 при переполнении разрядной сетки во время арифметических операций;
- биты 12-13 – текущий уровень привилегий для операций ввода-вывода.

1.5. Режимы работы процессора

«*Реальный*» *режим* устанавливается автоматически сразу после включения питания. Его характеристики:

- разрядность шины данных - 16 бит;
- разрядность шины адреса – 20 бит (т.е. процессору доступны адреса памяти от 0 до 2^{20}-1, всего 1024×1024=1Мб различных адресов);
- разрядность служебных регистров, регистров общего назначения и сегментных регистров – 16 бит.

«*Защищенный*» *режим* может быть запущен из «реального» после настройки служебных таблиц, необходимых для адресации. Характеристики защищенного режима:

- разрядность шины данных – 32 бита;

- разрядность шины адреса – 32 бита (т.е. процессору доступны 2^{32}=4Гб различных адресов) или 64 бита (т.е. процессору доступны 2^{64} различных адресов);
- разрядность служебных регистров и регистров общего назначения – 32 или 64 бита (сегментных – всегда 16 бит).

Все современные операционные системы работают только в «защищенном» режиме, а «реальный» режим моделируется по мере необходимости в виртуальных машинах типа DosBox или NTVDM.

1.6. Прерывания и исключения

Прерывание – реакция процессора на запрос (электрический импульс) от внешнего устройства (таймера, жесткого диска, интерфейса USB, видеоадаптера, сетевого адаптера, звуковой подсистемы и т.п.). Так же процессор может послать запрос прерывания сам себе при помощи специальной машинной команды «INT».

Исключение – реакция процессора на особое событие, происходящее внутри процессора при выполнении машинных команд. Примеры событий: ошибка деления на 0; ошибка при попытке доступа к «запрещенному» региону памяти; ошибка при попытке доступа к физически отсутствующему региону памяти и т.п.

Все источники прерываний и исключений пронумерованы операционной системой. Каждому номеру в специальной *таблице векторов прерываний* соответствует некоторый адрес системной памяти, по которому располагается *программа обработки прерывания/исключения*. Примеры номеров прерываний/исключений:

- 0 – ошибка деления на 0;
- 3 – «самомпрерывание»;
- 6 – недопустимый код команды;
- D_{16} – попытка обратиться в запрещенный регион памяти (например, по адресу 0);
- E_{16} – попытка обратиться в отсутствующий регион памяти.

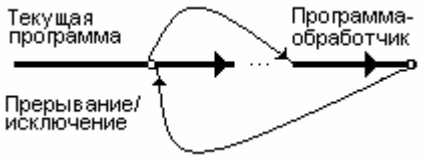

Рис. 4. Принцип обработки прерываний/исключений

При поступлении запроса на прерывание или при возникновении исключения процессор обязан прореагировать стандартным образом (см. рис. 4):

- сохранить в стеке регистр флагов;
- сохранить в стеке адрес очередной команды в текущей программе;
- перейти на выполнение программы-обработчика, адрес которой указан в таблице векторов прерываний.

Программа-обработчик прерывания/исключения должна заканчиваться специальной машинной командой «IRET». При выполнении этой команды процессор:

- восстанавливает из стека адрес очередной команды в текущей программе;
- восстанавливает из стека регистр флагов;
- продолжает выполнение прерванной программы с восстановленного адреса.

2. Некоторые ассемблерные команды

Для программирования процессоров Intel могут быть использованы различные языки ассемблера:

- с синтаксисом от Intel (используется в компиляторах для DOS и Windows);
- с синтаксисом, определенным в стандарте AT&T (используется в UNIX).

2.1. Команды пересылки данных

MOV приемник, источник –

пересылает данные из источника в приемник. Приемник и источник не могут одновременно быть ячейками памяти. Приемник и источник должны иметь одну и ту же разрядность.

LEA приемник, источник –

помещает в приемник адрес источника.

MOVSB или MOVSW или MOVSD –

пересылают данные из DS:[SI] в ES:[DI], автоматически увеличивая (или уменьшая) индексы SI и DI. Индексы увеличиваются, если в регистре флагов командой CLD предварительно сброшен бит D в 0; уменьшаются, если бит D установлен в 0 командой STD.

XCHG операнд1, операнд2 –

обменивает местами операнды. Операнды не могут одновременно быть ячейками памяти. Операнды должны иметь одну и ту же разрядность.

PUSH операнд –

значение указателя в стеке уменьшается: ESP:=ESP-(размер операнда). Помещает значение операнда в стек по адресу SS:[ESP]. Допустимая длина операнда: 2 или 4.

POP операнд –

извлекает значение из стека SS:[ESP] и помещает его в операнд. Указатель в стеке увеличивается на длину операнда ESP:=ESP+(размер операнда). Допустимая длина операнда: 2 или 4.

PUSHF –

работает аналогично PUSH, но помещает в стек не любой операнд, но регистр флагов.

POPF –

работает аналогично POP, но извлекает из стека не любой операнд, но регистр флагов.

PUSHA –

помещает в стек сразу весь блок регистров общего назначения.

POPA –

Извлекает из стека сразу весь блок регистров общего назначения.

2.2. Арифметические и логические команды

ADD операнд1, операнд2 –

Складывает операнды: операнд1:=операнд1+операнд2

SUB операнд1, операнд2 –

Вычитает операнды: операнд1:=операнд1-операнд2

ADC, SBB –

группа команд, аналогичных командам ADD и SUB, за исключением того, что к после вычисления результата к нему арифметически добавляется/вычитается значение бита CF из регистра флагов.

INC операнд –

Инкрементирует операнд: операнд:=операнд+1

DEC операнд –

Декрементирует операнд: операнд:=опеаранд-1

MUL операнд –

Умножает операнд на регистр AL, AX или EAX, помещает результат в AX, DX:AX или EDX:EAX, соответственно. Вид операции определяется длиной операнда, который может быть только регистром или ячейкой памяти (но не непосредственным числом).

DIV операнд –

Делит AL, AX или EAX на операнд, помещая результат в AL, AX или EAX и остаток в AH, DX или EDX, соответственно. Вид операции определяется длиной операнда, который может быть только регистром или ячейкой памяти (но не непосредственным числом).

IADD, ISUB, IMUL, IDIV –

группа арифметических команд, аналогичных командам ADD, SUB, MUL и DIV, за исключением того, что в результате не затрагивается старший (знаковый) бит операнда. В итоге арифметические действия производятся с учетом знака операндов.

AND операнд1, операнд2 –

Логически умножает операнды: опенранд1:=операнд1&опенранд2.

OR операнд1, операнд2 –

Логически складывает операнды: операнд1:=операнд1|операнд2.

XOR операнд1, операнд2 –

Выполняет операцию «сложение по модулю 2»: операнд1 := операнд1 ⊕ операнд2.

NOT операнд –

Инвертирует все биты в операнде.

NEG операнд –

Изменяет знак числа на противоположный.

CMP операнд1, операнд2 –

Выполняет сравнение двух операндов (путем арифметического вычитания), не изменяя их. Результат сравнения – установка в 1 или сброс в 0 отдельных битов в регистре флагов.

TEST операнд1, операнд2 –

Выполняет сравнение двух операндов (путем логического умножения), не изменяя их. Результат сравнения – установка в 1 или сброс в 0 отдельных битов в регистре флагов.

SHR операнд1,операнд2 и SHL операнд1,операнд2 –

выполняют сдвиг всех битов операнда1 вправо/влево на число позиций, указанное в операнде2. Последний «вытесненный» бит сохраняется в бите CF регистра флагов. «Новые» биты заполняются нулями.

ROR операнд1,операнд2 и ROL операнд1,операнд2 –

выполняют циклический сдвиг всех битов операнда1 вправо/влево на число позиций, указанное в операнде2.

RCR операнд1,операнд2 и RCL операнд1,операнд2 –

выполняют циклический сдвиг всех битов операнда1 вправо/влево на число позиций, указанное в операнде2. Сдвиг выполняется через бит CF в регистре флагов.

BT операнд, смещение –

помещает бит операнда, номер которого задан смещением, в бите CF регистра флагов.

BTS операнд, смещение –

устанавливает в 1 в операнде бит, номер которого задан смещением. Старое значение бита сохраняется в бите CF регистра флагов.

BTC операнд, смещение –

сбрасывает в 0 в операнде бит, номер которого задан смещением. Старое значение бита сохраняется в бите CF регистра флагов.

BTR операнд, смещение –

инвертирует в операнде бит, номер которого задан смещением. Старое значение бита сохраняется в бите CF регистра флагов.

BSF приемник, источник –

помещает в приемник номер первого по счету (начиная с младшего) бита 1 в источнике.

BSR приемник, источник –

Помещает в приемник номер первого по счету (начиная со старшего) бита 0 в источнике.

STC, STD, STI –

группа команд, которые устанавливают в 1 значения битов CF, DF и IF (соответственно) в регистре флагов.

CLC, CLD, CLI –

группа команд, которые сбрасывают в 0 значения битов CF, DF и IF (соответственно) в регистре флагов.

2.3. Команды перехода

JMP метка –

Выполняет безусловный переход по адресу, сопоставленному метке.

J* метка –

Выполняет переход по адресу, сопоставленному метке, если выполнено условие:

- JA – переход по «больше»;
- JB/JC – переход по «меньше» (если сброшен бит C);
- JAE/JNC – переход по «больше или равно» (если установлен бит C);
- JBE – переход по «меньше или равно»;

- JE/JZ – переход по «равно».

CALL метка –

Переходит на подпрограмму по адресу, сопоставленному метке. Предварительно адрес текущей команды автоматически помещается в стек.

RET –

Выполняет возврат из подпрограммы по адресу, извлеченному из стека. Так же существует разновидность «RET количество», которая после перехода удаляет из стека указанное количество байтов.

INT номер –

Возбуждает программное прерывание с указанным номером. При этом в стек помещаются регистр флагов и адрес текущей команды, затем выполняется переход по адресу, установленному для этого номера прерывания в системных таблицах процессора. Так же для возбуждения прерывания с номером 3 кроме «INT 3» существует «укороченная» команда «INT3», машинный код которой CC_{16} занимает один байт.

IRET –

Выполняет возврат из обработки прерывания по адресу, извлеченному из стека. Так же из стека извлекается регистр флагов.

LOOP метка –

Выполняет переход на адрес, сопоставленный метке, если ECX <> 0, уменьшая счетчик ECX на 1: ECX:=ECX-1; ничего не делает и переходит к следующей команде, если ECX=0. Если бит D установлен в 1, то команда каждый раз не уменьшает, а увеличивает ECX.

2.4. Прочие команды

IN AL, порт –

Принимает в регистр AL из порта ввода-вывода, связанного с каким-либо устройством, числовое значение.

OUT порт, AL –

Посылает в порт ввода-вывода, связанный с каким-либо устройством, числовое значение из регистра AL.

CPUID –

Возвращает в регистрах техническую информацию о процессоре, тип интересующей информации заранее задается в регистре EAX. Например, если EAX=0, то в EBX:ECX:EDX будет возвращена строка имени процессора.

RDTSC –

Возвращает в регистровой паре EDX:EAD количество тактовых импульсов, накопленное процессором.

3. Способы адресации операндов

Способы адресации определяют, каким образом в операнде команды может быть указан адрес регистра или ячейки памяти.

Регистровый. Операнд – регистр процессора. Пример: «NOT EAX».

Непосредственный. Операнд – число. Пример: «MOV EAX, 12345678h».

Прямой. Операнд – числовой адрес ячейки памяти. Пример: «MOV [12345678h], EAX».

Косвенный. Операнд – регистр, содержащий адрес ячейки памяти. Пример: «MOV [EBX], 0».

Косвенный со смещением. Операнд – комбинация регистра, содержащего адрес ячейки памяти, и числового смещения относительно этого адреса. Пример: «MOV [EBX+4], 0».

Базово-индексный. Операнд – комбинация регистра, содержащего адрес ячейки памяти, и регистра, содержащего смещение относительно этого адреса. Пример: «MOV [EBX+ESI], 12345678h».

При использовании косвенных и базово-индексных способов адресации часто требуется дополнительно указывать размер операнда при помощи модификаторов:

- byte ptr – байт (8 битов);
- word ptr – слово (16 битов);
- dword ptr – двойное слово (32 бита).

Примеры:

MOV BYTE PTR [EBX], AL – копировать 8 битов из регистра в память;

MOV EAX, DWORD PTR [EBX] – копировать 32 бита в регистр из памяти.

4. Отличия синтаксиса AT&T от Intel

Имена регистров в AT&T начинаются с символа «%», например:

- EAX – Intel;
- %EAX – AT&T.

Числовые константы в Intel имеют суффикс «h» для 16-ричных значений и не имеют суффикса для 10-чных. Числовые константы в AT&T оформляются по правилам языка Си и имеют префикс «$». Например:

- 123h – Intel;
- $0x123 – AT&T.

Команды AT&T, имеющие операнды, используют суффикс для указания размера операнда (-ов): «B» - один байт; «W» - слово; «D» - двойное слово; «Q» - четверное слово. Команды Intel могут для этих же целей (если один из

операндов находится в памяти) использовать ключевые слова «BYTE PTR», «WORD PTR», «DWORD PTR» и «QWORD PTR». Например:

- INC AL - Intel;
- INCB %AL – AT&T.

Если команда использует два операнда, то по правилам Intel сначала указывается приемник, потом источник; в синтаксисе AT&T наоборот. Например:

- MOV EAX, 123h – Intel;
- MOVD $0x123, EAX – AT&T.

Для прямой адресации в синтаксисе Intel используются квадратные скобки, в AT&T число или метка без суффиксов и префиксов. Пример:

- MOV AL, BYTE PTR [123456h] – Intel;
- MOVD 0x123456, %AL – AT&T.

Для доступа к адресу переменной в Intel используется ключевое слово «OFFSET», в AT&T просто имя метки. Пример:

- MOV EAX, OFFSET METKA - Intel;
- MOVD METKA, %EAX – AT&T.

5. Некоторые приемы программирования на языке ассемблера

5.1. Как организовать цикл

В простейшем случе можно воспользоваться в качестве счетчика любым регистром, например

```
      Mov eax, 123  ; Количество повторов
Oncemore:           ; Метка возврата
      …
      dec eax       ; Вычесть из eax единицу
      jnz Oncemore  ; Если результат предыдущей операции не 0, то на метку            …
                    ; Иначе дальше
```

Для организации циклов существует специальная команда LOOP, которая использует регистры ECX (или CX), автоматически уменьшая (или увеличивая) его содержимое на 1 и проверяя на равенство нулю. Если достигнут 0, то конец цикла.

```
      cld           ; Очистить бит D
      Mov ecx, 123  ; Количество повторов
Oncemore:           ; Метка возврата
      …
      loop oncemore ; Если ecx не 0, то на метку
      …             ; Иначе дальше
```

5.2. Как просканировать массив чисел или строку

В простейшем случае можно организовать индекс адреса в любом регистре и воспользоваться командой MOV, например

```
      ; Mov eax, seg STR  ; Не обязательные строки, чеще всего требуются
      ; Mov ds, eax       ; в MS-DOS, если сегментный регистр «испорчен».
      Mov ecx, 123        ; Количество повторов
      Lea edx, STR[0]     ; Адрес 1-го элемента массива в EDX
Oncemore:
      Mov eax, [edx]      ; Очередной элемент в EAX
      …                   ; Работа с очередным элементом
      Add edx, 4          ; Увеличение адреса
      Loop cx, oncemore   ; Возврат в цикл
```

Для сканирования и копирования массивов существуют специальные команды, использующие в качестве индекса регистры ESI и EDI:

- LODSB (или LODSW, или LODSD) - копирует в AL (или AX, или EAX) содержимое по адресу [ESI] и автоматически увеличивает или уменьшает ESI (в зависимости от значения бита D в регистре флагов);
- STOSB (или STOSW, или STOSD) – копирует из AL (или AX, или EAX) содержимое по адресу [EDI] и автоматически увеличивает или уменьшает EDI (в зависимости от значения бита D в регистре флагов);
- MOVSB (или MOVSW, или MOVSD) – копирует [ESI] в [EDI] и автоматически увеличивает или уменьшает ESI и EDI (в зависимости от значения бита D в регистре флагов).

```
      Cld                 ; Бит D:=1, т.е. каждый раз увеличивать ESI
      Mov ecx, 123        ; Количество повторов
      Lea esi, Str[0]     ; Адрес 1-го элемента массива
Oncemore:
      Lodsd               ; Очередной элемент загрузить в EAX, при этом ESI:=ESI+4
      …                   ;
      Loop cx, oncemore   ; Возврат в цикл
```

5.3. Как сложить или вычесть два числа

Легко!

```
Mov eax, 123
Mov ebx, 321
Add eax, ebx ; eax:=eax+ebx
```

5.4. Как разделить или умножить число на степень двойки

Посольку числа хранятся в 2-чном коде, то умножение на 2 эквивалентно сдвигу всех битов влево на один разряд. Соответственно, деление – сдвигу вправо.

```
Mov eax, 123 ; Само число
shl eax, 3   ; Теперь в EAX появится 123*8=984
```

Вновь появившиеся разряды заполняются нулями. При сдвиге влево последний «выпавший» бит помещается в бит O регистра флагов, при сдвиге вправо – в бит C.

5.5. Как поменять местами два операнда

В простейшем случае через «битончик»:

```
Mov eax, 123 ; Первое число
Mov ebx, 321 ; Второе число
Mov ecx, eax ; В битончик
Mov eax, ebx
Mov ebx, ecx ; Из битончика
```

Можно через стек:

```
Push eax
Push ebx
Pop eax
Pop ebx
```

Можно специальной командой:

```
Xchg eax, ebx
```

Ну а можно воспользоваться свойствами логической функции «исключающее ИЛИ»:

```
Xor eax, ebx
Xor ebx, eax
Xor eax, ebx
```

5.6. Как проверить конкретный бит операнда

Если проверить на 0, то:

```
Tst al, 00010000b
Je Metka
```

Если проверить на 1, то:

```
Tst al, 00010000b
Jne Metka
```

5.7. Как установить в 1 или сбросить в 0 конкретный бит операнда

Если сбросить в 0, то:

```
And al, 11101111b
```

Если установить в 1, то

```
Or al, 00010000b
```

5.8. Как обратиться к операционной системе

В операционной системе MS Windows загрузить параметры в стек в обратном порядке и вызвать библиотечную функцию Win API командой «CALL».

```
Push 0                  ; 0 – кнопка "OK"
Push offset String1     ; Адрес строки заголовка
Push offset String2     ; Адрес строки сообщения
Push 0                  ; Дескриптор окна
Call MessageBoxA        ; Если строки ASCII
      или
Call MessageBoxW        ; Если строки UNICODE
```

Некоторые компиляторы поддерживают директивы вида «INVOKE 0,String2, String2,0», которые при компиляции автоматически «расширяются» в вышеприведенный текст из 5 команд.

В операционной системе на базе ядра Linux поместить параметры в регистры (в том числе и номер API-функции) и выполнить вызов командой «INT 80h».

```
movl $4, %eax  ; Номер функции: write=4, read=3, open=5, fork=2, exit=1...
movl $1, %ebx  ; Дескриптор файла: stdin=0, stdout=1, stderr=2...
movl $str, %ecx ; Адрес строки, кончающейся кодом 0
movl $6, %edx  ; Длина строки
int 0x80
```

В операционной системе MS-DOS поместить параметры в регистры (в том числе и номер API-функции) и выполнить вызов командой «INT 21h».

```
mov ax, 9                ; Номер функции «вывод на экран»
mov dx, offset String    ; Адрес строки, кончающейся символом «$»
int 21h
```

6. Примеры программ

6.1. Сортировка массива 4-байтовых чисел

```
/* Синтаксис Intel */
#include <stdio.h>
unsigned long a[5]={30, 10, 50, 20, 40};
int i;
main() {
 for (i=0;i<5;i++) printf(" %d", a[i]);
 asm {
         mov  edx, 4
oncemore: mov  ecx, edx
         mov  ebx, offset a
repeat:   mov  eax, dword ptr [ebx]
         cmp  eax, [ebx+4]
         jle  skip
          xchg [ebx+4], eax
          mov  dword ptr [ebx], eax
skip:     add  ebx, 4
          loop repeat
         dec  edx
          jnz  oncemore
}
 printf("\n");
 for (i=0;i<5;i++) printf(" %u", a[i]);
}
```

```
/*Синтаксис AT&T */
#include <stdio.h>
unsigned a[5]={30, 10, 50, 20, 40};
int i;
main() {
 for (i=0;i<5;i++) printf(" %d", a[i]);

 asm("          movl  $0x4, %edx");
 asm("oncemore: movl  %edx, %ecx");
 asm("          movl  $a, %ebx");
 asm("repeat:   movl  (%ebx), %eax");
 asm("          cmpl  0x4(%ebx), %eax");
 asm("          jle   skip");
 asm("          xchgl 0x4(%ebx), %eax");
 asm("          movl  %eax, (%ebx)");
 asm("skip:     addl  $0x4, %ebx");
 asm("          loop  repeat");
 asm("          decl  %edx");
 asm("          jnz   oncemore");

 printf("\n");
 for (i=0;i<5;i++) printf(" %d", a[i]);
}
```

6.2. Вывод на экран через обращение к ОС

```
#include <windows.h>
#include <stdio.h>
char ProcName[]="0___4___8___\0"; // Сюда пойдет имя процессора
char Title[]="Attention!!!";      // Это заголовок окна
char Message[32];                 // Сюда пойдет вся строка
unsigned char Model;              // Здесь в битах 0..3 номер семейства
main() {
 asm {
  .586
  mov eax,0
```

```
  cpuid
  mov dword ptr [ProcName+0], ebx
  mov dword ptr [ProcName+4], ecx
  mov dword ptr [ProcName+8], edx
  mov eax,1
  cpuid
  and ah, 00001111b
  mov Model, ah
 }
 sprintf(Message,"My processor is %s 80x%d86", ProcName, Model);
 asm {
  push dword ptr 0
  push offset Title[0]
  push offset Message[0]
  push dword ptr 0
  call MessageBoxA
 }
}
```

Литература

1. Зубков С.В. Assembler для DOS, Windows и UNIX. – М.: ДМК, 1999. – 640 с.

2. Аблязов Р.З. Программирование на ассемблере на платформе x86-64. – М.: ДМК, 2011. – 304 с.

3. Магда Ю. Ассемблер для процессоров Intel Pentium. – М.: Питер, 2006. - 448 с.

Приложение А. Варианты индивидуальных заданий

1. Переставить все числа массива в обратном порядке
2. Смоделировать работу LFSR (суммируемые биты выбрать любые)
3. Рассчитать полусумму максимума и минимума в массиве чисел
4. Рассчитать разность максимума и минимума в массиве чисел
5. Подсчитать количество гласных букв в строке
6. Подсчитать количество согласных букв в строке
7. Подсчитать количество четных неотрицательных чисел в массиве
8. Подсчитать количество нечетных отрицательных чисел в массиве
9. Зашифровать строку символов методом Цезаря
10. Посчитать, сколько раз в числе встречается сочетание «101»
11. Проверить целое число на простоту методом brute force
12. Заменить в русской строке похожие буквы латиницей
13. Заменить в латинской строке похожие буквы кириллицей
14. Сравнить строки на совпадение путем простых контрольных сумм
15. Рассчитать бит четности для операнда
16. Преобразовать строку 2-чных цифр в целое число
17. Преобразовать целое число в строку 2-чных цифр
18. Преобразовать строку 10-чных цифр в строку
19. Преобразовать целое число в строку 10-чных цифр
20. Подсчитать количество битов 1 в числе
21. Подсчитать количество битов 0 в числе
22. Расставить три числа по порядку без сортировки
23. Посчитать, сколько раз в числе встречается сочетание «01»

24. Поменять местами максимальное и минимальное числа в массиве
25. Поменять местами в числе соседние биты 0 и 1, 2 и 3, 4 и 5 и т.д.

Приложение Б. Справочное

Шифр Цезаря основан на замене по принципу C←(C+K) mod |A|, где C– номер в алфавите для буквы сообщения (см. рис. 5), K – номер буквы ключа, |A| - мощность алфавита. Пусть слово «НЕКИЙ» шифруется ключом «Е», тогда результат «УКРОП».

00-01-02-03-04-05-06-07-08-09-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32
А Б В Г Д Е Ё Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Ь Э Ю Я
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Рис. 5. Номера букв в алфавите

Шифр Вижинера подобен шифру Цезаря, только каждая буква шифруется отдельной буквой ключа-строки. Например, слово «БАТАТ» с ключом «СЕЯТЬ» зашифруется как «ТЕСТО».

Бит четности (бит паритета) – сумма по модулю 2 всех битов блока данных. Например, $37_{16}=00110111_2$, тогда бит паритета равен $0\oplus0\oplus1\oplus1\oplus0\oplus1\oplus1\oplus1=1$.

LFSR (или РСЛОС) – регистр сдвига с линейной обратной связью, способ генерации псевдослучайной последовательности битов. Вначале регистр инициализируется начальным значением (вектором инициализации). На каждом очередном шаге генерации некоторые, заранее выбранные (например, с номерами 1, 2 и 6) биты регистра складываются по модулю 2. Затем содержимое регистра сдвигается на одну позицию, «опустевший» бит заменяется результатом сложения, а «вытесненный» бит служит очередным битом псевдослучайной последовательности (см. рис. 6).

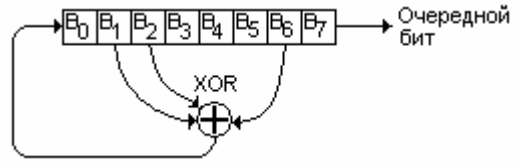

Рис. 6. Работа LFSR

Простые контрольные суммы:

- сумма всех элементов (например, байтов) набора данных;
- сумма по модулю 2 всех элементов (например, байтов) набора данных;
- (сумма Бернштейна, Bernstein hash, Djb2) взвешенная сумма всех элементов (например, байтов) набора данных: $S\leftarrow S\times A+C_i$, здесь S – контрольная сумма, A – простое число (например, 31), C_i – очередной элемент последовательности;
- взвешенная сумма по модулю 2 всех элементов (например, байтов) набора данных.