Лабораторная работа № 3_1

Выполнение арифметических операций над числами без знака и со знаком

Цель работы

Научиться выполнять операции сложения, вычитания, умножения и деления с целыми числами без знака и со знаком.

Порядок выполнения работы

1. Создать рабочую папку для текстов программ на ассемблере и записать в нее файлы tasm.exe, tlink.exe, rtm.exe и td.exe из пакета tasm, а также файл с исходным текстом программы на ассемблере, который сохранить с именем prog4.asm.
2. Создать загрузочный модуль, загрузить его в отладчик и выполнить программу в пошаговом режиме.

Содержание отчета:

1. Цель работы.
2. Постановка задачи.
3. Листинг программы.
4. Окна **Dump** отладчика для **PRG_№.exe**, с отмеченными результатами арифметических операций.
5. Таблица изменения состояния регистров при выполнении программы.
6. Вывод.

Постановка задачи:

Задание 1

Разработать программу для каждого случая согласно индивидуальному варианту заданию при решении следующей задачи.

1.1) В сегменте данных определить:

- байтовые значения Xi, a, b, c, d в десятичной системе счисления;

1.2) В сегменте данных зарезервировать байтовые ячейки для хранения:

- суммы и разности с нулевыми первоначальными значениями,
- двухбайтовую ячейку для хранения произведения с единичным первоначальным значением,
- две байтовые ячейки для хранения остатка от деления и частного с произвольными первоначальными значениями.

1.3) Написать программу на языке Ассемблер, которая выполняет следующие операции над байтовыми значениями.

- из каждого целого числа Xi, где (i = 1, 2, 3), требуется вычесть число a ,
- к результату добавить число b,
- полученный результат умножить на число c,
- полученный результат разделить на число d.
- Определить сумму инкремента неполного частного e и декремента остатка от деления f.

1.4.) Результаты работы программ вместе с их листингами внести в отчет и сравнить со значениями, полученными при выполнении тех же вычислений вручную.

Таблица вариантов

№ варианта	Значения переменных						
	X_1	X_2	X_3	A	B	C	D
1	76	7Bh	245	45	10h	12	5
2	45	84h	223	25	0Eh	10	9
3	56	A1h	232	51	15h	11	8
4	75	8Ah	227	34	20h	15	7
5	68	81h	252	29	13h	13	12
6	84	9Ch	217	48	22h	4	4
7	59	8Dh	245	44	29h	7	3
8	92	A1h	222	39	33h	5	6
9	85	84h	231	77	4Eh	8	7
10	72	A3h	218	62	3Eh	12	8
11	84	B8h	226	45	48h	10	3
12	97	6Eh	213	55	26h	3	5
13	79	81h	222	39	38h	2	2
14	85	97h	231	62	31h	6	8
15	99	B6h	201	21	36h	7	2
16	56	C5h	214	12	48h	3	9

Теоретическая часть

Микропроцессор выполняет сложение операндов по правилам сложения двоичных чисел. В микропроцессоре этот исход сложения прогнозируется и предусмотрены специальные средства для фиксирования подобных ситуаций и их обработки. Итак, для фиксирования ситуации выхода за разрядную сетку результата, используются флаги. В системе команд микропроцессора имеются три команды двоичного сложения - ADD, ADC, INC.

Рассмотрим синтаксис команды ADD.

ADD (ADDition)

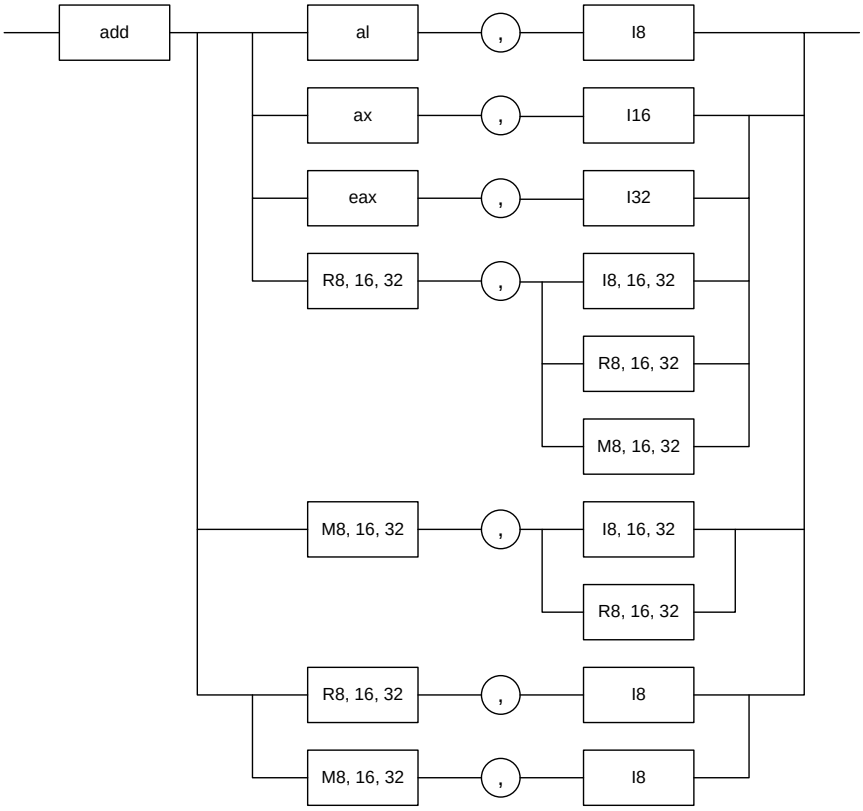
Сложение

```
------------------------------------------------
            add     приемник, источник
------------------------------------------------
```

Назначение:

Сложение двух операндов *источник* и *приемник* размерностью байт, слово или двойное слово.

Синтаксис:



Синтаксическое описание команды **add**

Алгоритм работы:

- сложить операнды *источник* и *приемник*;
- записать результат сложения в *приемник*;
- установить флаги;

add операнд_1, операнд_2 — команда сложения с принципом действия:
операнд_1 = операнд_1 + операнд_2

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:

Команда **add** используется для сложения двух целочисленных операндов. Результат сложения помещается по адресу первого операнда. Если результат сложения выходит за границу операнда приемник (возникает переполнение), то учесть эту ситуацию следует путем анализа флага cf и последующего возможного применения команды adc.

Например, сложим значения в регистре ax и области памяти ch. При сложении следует учесть возможность переполнения:

```
Chislo     dw    2015
Rez        dd    0

add   ax, chislo            ;(ax)=(ax)+ch
mov   word ptr rez, ax
jnc   dop_sum               ;переход, если результат
                            ;не вышел за разрядную сетку
adc   word ptr rez+2, 0     ;расширить результат,
                            ;для учета переноса в старший разряд
dop_sum:
...
```

SUB (SUBtract)

Вычитание.

```
-------------------------------------------------------
           sub    операнд_1,    операнд_2
-------------------------------------------------------
```

Назначение:

Целочисленное вычитание.

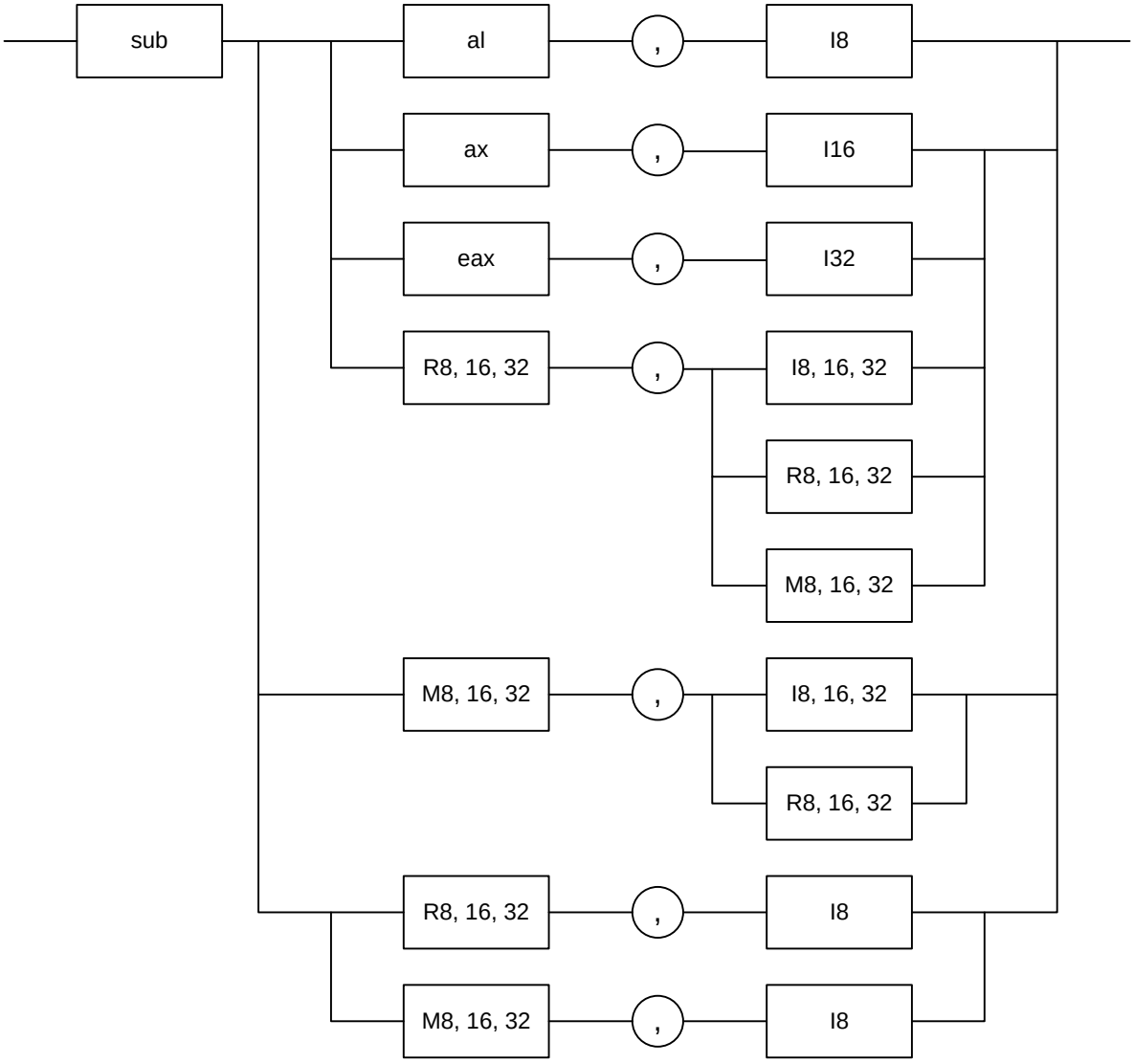
Алгоритм работы:

- выполнить вычитание операнд_1=операнд_1-операнд_2;
- установить флаги.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Синтаксис:



Синтаксическое описание команды **sub**

Применение:

Команда **sub** используется для выполнения вычитания целочисленных операндов или для вычитания младших частей значений многобайтных операндов:

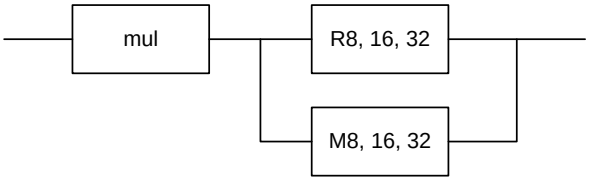
```
; выполнить вычитание 64-битных значений: vich_1-vich_2
vich_1     dd    2 dup (0)
vich_2     dd    2 dup (0)
rez        dd    2 dup (0)
...
;ввести значение в поля vich_1 и vich_2:
;младший байт по младшему адресу
...
mov   eax, vich_1
sub   eax, vich_2          ;вычесть младшие половинки чисел
mov   rez, eax        ;младшая часть результата
mov   eax, vich_1+4
sbb   eax, vich_2+4   ;вычесть старшие половинки чисел
mov   rez+4, eax      ;старшая часть результата
```

MUL(MULtiply)
Умножение целочисленное без учета знака.

```
-----------------------------------------------
            mul       множитель_1
-----------------------------------------------
```

Назначение:
Операция умножения двух целых чисел без учета знака.

Синтаксис:



Синтаксическое описание команды **mul**

Алгоритм работы:

Команда выполняет умножение двух операндов без учета знаков. Алгоритм зависит от формата операнда команды и требует явного указания местоположения только одного сомножителя, который может быть расположен в памяти или в регистре. Местоположение второго сомножителя фиксировано и зависит от размера первого сомножителя:

- если операнд, указанный в команде, - байт, то второй сомножитель должен располагаться в al;
- если операнд, указанный в команде, - слово, то второй сомножитель должен располагаться в ax;
- если операнд, указанный в команде, - двойное слово, то второй сомножитель должен располагаться в eax.

Результат умножения также помещается в фиксированное место, определяемое размером сомножителей:

- при умножении байтов результат помещается в ax;
- при умножении слов результат помещается в пару dx:eax;
- при умножении двойных слов результат помещается в пару edx:eax.

Состояние флагов после выполнения команды (если старшая половина результата нулевая):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
0	?	?	?	?	0

Состояние флагов после выполнения команды (если старшая половина результата ненулевая):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
1	?	?	?	?	1

Применение:

Команда **mul** выполнение целочисленное умножение операндов без учета их знаковых разрядов. Для этой операции необходимо наличие двух операндов-сомножителей, размещение одного из которых фиксировано, а другого – задается операндом в команде. Контролировать размер результата удобно, используя флаги cf и of.

```
mn_1  db    15
mn_2  db    25
...
```

```
mov   al, mn_1
mul   mn_2
```

IMUL(Integer MULtiply)

Схема команды:	imul множитель_1 imul множитель_1, множитель_2 imul результат, множитель_1, множитель_2

Умножение целочисленное со знаком
Назначение*:* операция умножения двух целочисленных двоичных значений со знаком.

Алгоритм работы:
Алгоритм работы команды зависит от используемой формы команды. Форма команды с одним операндом требует явного указания местоположения только одного сомножителя, который может быть расположен в ячейке памяти или регистре. Местоположение второго сомножителя фиксировано и зависит от размера первого сомножителя:

- если операнд, указанный в команде, — байт, то второй сомножитель располагается в al;
- если операнд, указанный в команде, — слово, то второй сомножитель располагается в ax;
- если операнд, указанный в команде, — двойное слово, то второй сомножитель располагается в eax.

Результат умножения для команды с одним операндом также помещается в строго определенное место, определяемое размером сомножителей:

- при умножении байтов результат помещается в ax;
- при умножении слов результат помещается в пару dx:ax;
- при умножении двойных слов результат помещается в пару edx:eax.

Команды с двумя и тремя операндами однозначно определяют расположение результата и сомножителей следующим образом:

- в команде с двумя операндами первый операнд определяет местоположение первого сомножителя. На его место впоследствии будет записан результат. Второй операнд определяет местоположение второго сомножителя;
- в команде с тремя операндами первый операнд определяет местоположение результата, второй операнд — местоположение первого сомножителя, третий операнд может быть непосредственно заданным значением размером в байт, слово или двойное слово.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	?	?	?	?	r

Команда imul устанавливает в ноль флаги of и cf, если размер результата соответствует регистру назначения. Если эти флаги отличны от нуля, то это означает, что результат слишком велик для отведенных ему регистром назначения рамок и необходимо указать больший по

размеру регистр для успешного завершения данной операции умножения. Конкретными условиями сброса флагов of и cf в ноль являются следующие условия:

- для одноперандной формы команды imul регистры ax/dx/edx являются знаковыми расширениями регистров al/ax/eax;
- для двухоперандной формы команды imul для размещения результата умножения достаточно размерности указанных регистров назначения r16/r32;
- то же для трехоперандной команды умножения.

Применение:

Команда выполняет целочисленное умножение операндов с учетом их знаковых разрядов. Для выполнения этой операции необходимо наличие двух сомножителей. Размещение и задание их местоположения в команде зависит от формы применяемой команды умножения, которая, в свою очередь, определяется моделью микропроцессора. Так, для микропроцессора i8086 возможна только одноперандная форма команды, для последующих моделей микропроцессоров дополнительно можно использовать двух- и трехоперандные формы этой команды.

Пример 1. Операция с 8-разрядными числами: -4 * 4 = -16

```
mov al, -4
mov bl, 4
imul bl         ;AX = FFF0h (-16), CF = 0, OF = 0
```

Результирующим значением в AX является число FFF0h (-16), и регистр AH получает знаковое расширение регистра AL (заполняется знаком AL), поэтому CF = 0 и OF = 0.

Пример 2. Операции с 16-разрядными числами: 48 * 4 = 192

```
mov ax, 48
mov bx, 4
imul bx          ; DX = 0000, AX = 00C0h (+192), CF = 0, OF = 0
```

Результат в DX:AX представляет 000000C0h. Так, знаки DX и AX являются одинаковыми (положительными), а CF = 0 и OF = 0.

Пример 3.

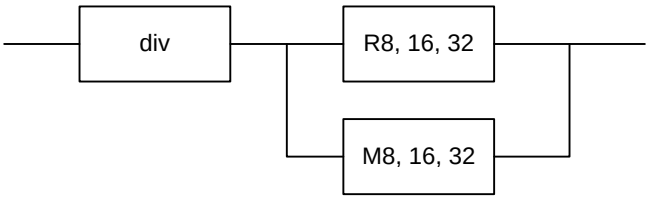
```
.486
...
        mov     bx,186
        imul    eax,bx,8
;если результату не хватило размерности операнда1,
;то перейдем на m1, где скорректируем ситуацию:
       jc      m1
```

DIV(DIVide unsigned)
Деление беззнаковое.

div	**делитель**

Назначение:
Выполнение операции деления двух двоичных беззнаковых значений.

Синтаксис:



Синтаксическое описание команды **div**

Алгоритм работы:
Для команды необходимо задание двух операндов – делимого и делителя. Делимое задается неявно, и размер его зависит от размера делителя, который указывается в команде:

- если делитель размером в байт, то делимое должно быть расположено в регистре ax. После операции частное помещается в al, а остаток – в ah;
- если делитель размером в слово, то делимое должно быть расположено в паре регистров dx:ax, причем младшая часть делимого находиться в ax. После операции частное помещается в ax, а остаток в dx;
- если делитель размером в двойное слово, то делимое должно быть расположено в паре регистров edx:eax, причем младшая часть делимого находиться в eax. После операций частное помещается в eax, а остаток в edx.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	?	?	?	?	?

Применение:
Команда выполняет целочисленное деление операндов с выдачей результата деления в виде частного и остатка от деления. При выполнении операции деление возможно возникновение исключительной ситуации 0 – ошибка деления. Эта ситуация возникает в одном из двух случаев: делитель равен 0 или частное слишком велико для его размещения в регистре eax/ax/al:

```
mov   ax, 10234
mov   bl, 154
div   bl                ; ah=остаток,  al=частное
```

IDIV (Integer DIVide)
Знаковое деление

idiv	**делитель**

Назначение: операция деления двух двоичных значений со знаком.

Алгоритм работы:

Для команды необходимо задание двух операндов — делимого и делителя. Делимое задается неявно, и размер его зависит от размера делителя, местонахождение которого указывается в команде:

- если делитель размером в байт, то делимое должно быть расположено в регистре ax. После операции частное помещается в al, а остаток — в ah;
- если делитель размером в слово, то делимое должно быть расположено в паре регистров dx:ax, причем младшая часть делимого находится в ax. После операции частное помещается в ax, а остаток — в dx;
- если делитель размером в двойное слово, то делимое должно быть расположено в паре регистров edx:eax, причем младшая часть делимого находится в eax. После операции частное помещается в eax, а остаток — в edx;

Остаток всегда имеет знак делимого. Знак частного зависит от состояния знаковых битов (старших разрядов) делимого и делителя.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	?	?	?	?	?

Применение:

Команда выполняет целочисленное деление операндов с учетом их знаковых разрядов. Результатом деления являются частное и остаток от деления. При выполнении операции деления возможно возникновение исключительной ситуации: 0 — ошибка деления. Эта ситуация возникает в одном из двух случаев: делитель равен 0 или частное слишком велико для его размещения в регистре eax/ax/al.

Применение:

Пример 1.

```
mov ax, -48     ;AX = FFD0h
mov bl, 5
idiv bl         ;AX = 0329h (частное = 41, остаток = 3)
```

Замечание: в случае если поместить 8 – разрядное делимое в регистр AL, то возможно возникновение ошибки, и частное будет неправильным

Пример 2. Деление слов

```
mov     ax,1045 ;делимое
mov     bx,587  ;делитель
cwd             ;расширение делимого dx:ax
idiv    bx      ;частное в ax, остаток в dx
```

NEG

Изменение знака

NEG op1 (op1 - r/m8, r/m16, r/m32)

Алгоритм работы:
Команда NEG выполняет над числом, содержащимся в операнде, операцию дополнения до двух. Значение операнда вычитается из нуля, а результат помещается обратно в операнд. Эта операция эквивалентна обращению знака операнда, если рассматривать его как число со знаком.
Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
? ?	?	?	?	?	

Применение:

Команда NEG позволяет инвертировать знак числа. Всегда трактует значение операнда, как число со знаком.

```
Пример 1.
neg     ax  ;AX = -AX
```

Команды DEC, INC
Рубрика: Архитектура PC (Команды CPU)
261
Опубликовано: 27.04.2017
Команды DEC и INC позволяют увеличивать или уменьшать значение операнда на 1. После выполнения операции флаговый регистр устанавливается в соответствии с результатом.

Команда DEC

Синтаксис:	**DEC op1**
Операнды:	**op1** - r/m8, r/m16, r/m32
Назначение:	Декремент
Процессор:	8086+
Флаги:	Флаги OF, SF, ZF, AF и PF устанавливаются в соответствии с результатом. Флаг CF не изменяется.
Комментарий:	Команда DEC уменьшает значение операнда на 1
Ограничения:	Нет
Примеры:	dec word ptr [bx]

Команда INC

Синтаксис:	**INC op1**
Операнды:	**op1** - r/m8, r/m16, r/m32
Назначение:	Инкремент
Процессор:	8086+
Флаги:	Флаги OF, SF, ZF, AF и PF устанавливаются в соответствии с результатом. Флаг CF не изменяется.

```
Комментарий:        Команда INC увеличивает значение операнда на 1
Ограничения:        Нет
Примеры:            inc     ax
```