Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОССУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2

По дисциплине «Архитектуры вычислительных систем»

По теме «Арифметическая операция умножения с целыми числами»

Выполнил:

студент гр. 653505

Хлопцев А. А.

Проверил:

Шиманский В. В.

Минск 2018

Оглавление

[1 Цель работы 3](#_Toc507669317)

[2 Постановка задачи 4](#_Toc507669318)

[3 Краткие теоретические сведения 5](#_Toc507669319)

[3.1 Представление чисел в компьютере 5](#_Toc507669320)

[4 Команды Assembler 8](#_Toc507669321)

[4.1 MOV 8](#_Toc507669322)

[4.2 MUL 9](#_Toc507669323)

[Пример умножения в Ассемблере 10](#_Toc507669324)

[5 Флаги 12](#_Toc507669325)

[6 Демонстрация работы программы 13](#_Toc507669326)

[7 Вывод 15](#_Toc507669327)

# Цель работы

Изучить как производится арифметическая операция умножения над целыми числами в языке ассемблер.

# Постановка задачи

Написать программу эмулятора АЛУ, реализующего операцию умножения с фиксированной точкой над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

# Краткие теоретические сведения

## Представление чисел в компьютере

**Представление чисел в формате с фиксированной запятой.** Целые числа в компьютере хранятся в памяти в формате с *фиксированной запятой*. В этом случае каждому разряду ячейки памяти соответствует всегда один и тот же разряд числа, а "запятая" "находится" справа после младшего разряда, то есть вне разрядной сетки.

Для хранения *целых неотрицательных чисел* отводится одна ячейка памяти (8 битов). Например, число А2 = = 111100002 будет храниться в ячейке памяти следующим образом:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Максимальное значение целого неотрицательного числа достигается в случае, когда во всех ячейках хранятся единицы. Для n-разрядного представления оно будет равно

2n - 1.

Определим диапазон чисел, которые могут храниться в оперативной памяти в формате *целых неотрицательных чисел*. Минимальное число соответствует восьми нулям, хранящимся в восьми битах ячейки памяти, и равно нулю. Максимальное число соответствует восьми единицам и равно

А = 1 × 27 + 1 × 26 + 1 × 25 + 1 × 24 + 1 × 23 + 1 × 22 + 1 × 21 + 1 × 20 = 1 × 28 - 1 = 25510.

Диапазон изменения *целых неотрицательных чисел* чисел: от 0 до 255.

Для хранения *целых чисел со знаком* отводится две ячейки памяти (16 битов), причем старший (левый) разряд отводится под знак числа (если число положительное, то в знаковый разряд записывается 0, если число отрицательное - 1).

Представление в компьютере положительных чисел с использованием формата "знак-величина" называется *прямым кодом* числа. Например, число 200210 = 111110100102 будет представлено в 16-разрядном представлении следующим образом:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

Максимальное положительное число (с учетом выделения одного разряда на знак) для целых чисел со знаком в n-разрядном представлении равно:

А = 2n-1 - 1.

Для представления отрицательных чисел используется *дополнительный код*. Дополнительный код позволяет заменить арифметическую операцию вычитания операцией сложения, что существенно упрощает работу процессора и увеличивает его быстродействие.

*Дополнительный код отрицательного числа А, хранящегося в n ячейках, равен 2n - |A|.*

Дополнительный код представляет собой дополнение модуля отрицательного числа А до 0, так как в n-разрядной компьютерной арифметике:

2n - |А| + |А| = 0,

поскольку в компьютерной n-разрядной арифметике 2n = 0. Действительно, двоичная запись такого числа состоит из одной единицы и n нулей, а в n-разрядную ячейку может уместиться только n младших разрядов, то есть n нулей.

Для получения дополнительного кода отрицательного числа можно использовать довольно простой алгоритм:

1. Модуль числа записать в *прямом коде* в n двоичных разрядах.

2. Получить *обратный код* числа, для этого значения всех битов инвертировать (все единицы заменить на нули и все нули заменить на единицы).

3. К полученному обратному коду прибавить единицу.

Запишем дополнительный код отрицательного числа -2002 для 16-разрядного компьютерного представления:

|  |
| --- |
| http://www.5byte.ru/11/images/inf14.gif |

При n-разрядном представлении отрицательного числа А в дополнительным коде старший разряд выделяется для хранения знака числа (единицы). В остальных разрядах записывается положительное число

2n-1 - |А|.

Чтобы число было положительным, должно выполняться условие

|А| £ 2n-1 .

Следовательно, максимальное значение модуля числа А в га-разрядном представлении равно:

|А| = 2n-1 .

Тогда минимальное отрицательное число равно:

А = - 2n-1.

Определим диапазон чисел, которые могут храниться в оперативной памяти в формате *длинных целых чисел со знаком* (для хранения таких чисел отводится четыре ячейки памяти - 32 бита).

Максимальное положительное целое число (с учетом выделения одного разряда на знак) равно:

А = 231 - 1 = 2 147 483 64710.

Минимальное отрицательное целое число равно:

А = -231 = - 2 147 483 64810.

# Команды Assembler

## MOV

Пожалуй, инструкция MOV в ассемблере самая простая. Синтаксис этой команды такой:

MOV ПРИЁМНИК, ИСТОЧНИК

С помощью этой команды можно переместить значение из ИСТОЧНИКА в ПРИЁМНИК. То есть по сути команда **MOV** копирует содержимое ИСТОЧНИКА и помещает это содержимое в ПРИЁМНИК.

Никакие флаги при этом НЕ изменяются.

При использовании этой команды следует учитывать, что имеются некоторые ограничения. А именно, инструкция MOV **не может**:

* Записывать данные в регистры CS и IP.
* Копировать данные из одного сегментного регистра в другой сегментный регистр (сначала нужно скопировать данные в регистр общего назначения).
* Копировать непосредственное значение в сегментный регистр (сначала нужно скопировать данные в регистр общего назначения).

ИСТОЧНИКОМ может быть один из следующих:

* Область памяти (MEM)
* Регистр общего назначения (REG)
* Непосредственное значение (например, число) (IMM)
* Сегментный регистр (SREG)

ПРИЁМНИКОМ может быть один из следующих:

* Область памяти (MEM)
* Регистр общего назначения (REG)
* Сегментный регистр (SREG)

С учётом ограничений, которые были описаны выше, комбинации ПРИЁМНИК-ИСТОЧНИК могут быть следующими:

REG, MEM

SREG, MEM

MEM, REG

REG, REG

SREG, REG

MEM, IMM

REG, IMM

MEM, SREG

REG, SREG

Пример использования инструкции MOV:

MOV AX, 0B800h ; установить AX = B800h (память VGA).

MOV DS, AX ; копировать значение из AX в DS.

MOV CL, 'A' ; CL = 41h (ASCII-код).

MOV CH, 01001110b ; CH = атрибуты цвета (желтый текст на красном фоне).

MOV BX, 72eh ; BX = позиция на экране = 2\*(x + y\*80).

MOV [BX], CX ; [0B800h:015Eh] = CX.

## MUL

**Инструкция MUL** в Ассемблере выполняет умножение без знака. Понять работу команды MUL несколько сложнее, чем это было для [**команд**](http://av-assembler.ru/instructions/), рассмотренных ранее. Но, надеюсь, что я помогу вам в этом разобраться.

Итак, синтаксис команды MUL такой:

MUL ЧИСЛО

Выглядит всё очень просто. Однако эта простота обманчива.

Прежде чем разобраться в подробностях работы этой инструкции, давайте посмотрим, что может быть ЧИСЛОМ.

ЧИСЛОМ может быть один из следующих:

* Область памяти (MEM)
* Регистр общего назначения (REG)

Эта команда не работает с сегментными регистрами, а также не работает непосредственно с числами. То есть вот так

MUL 200 ; неправильно

делать нельзя.

А теперь алгоритм работы команды MUL:

* Если ЧИСЛО - это БАЙТ, то AX = AL \* ЧИСЛО
* Если ЧИСЛО - это СЛОВО, то (DX AX) = AX \* ЧИСЛО

Вот такая немного сложная команда. Хотя сложно это с непривычки. Сейчас мы разберём всё “по косточкам” и всё станет ясно.

Для начала обратите внимание, что инструкция MUL работает либо с регистром АХ, либо с регистром AL. То есть перед выполнением этой команды нам надо записать в регистр АХ или в регистр AL значение, которое будет участвовать в умножении. Сделать это можно, например, с помощью уже известной нам [**команды MOV**](http://av-assembler.ru/instructions/mov.php).

Затем мы выполняем умножение, и получаем результат либо в регистр АХ (если ЧИСЛО - это байт), либо в пару регистров DX и AX (если ЧИСЛО - это слово). Причём в последнем случае в регистре DX будет старшее слово, а в регистре AX - младшее.

## Пример умножения в Ассемблере

Итак, например, нам надо умножить 150 на 250. Тогда мы делаем так:

MOV AL, 150 ; Первый множитель в регистр AL

MOV BL, 250 ; Второй множитель в регистр BL

**MUL** BL ; Теперь АХ = 150 \* 250 = 37500

Обратите внимание, что нам приходится два раза использовать команду MOV, так как команда MUL не работает непосредственно с числами, а только с регистрами общего назначения или с памятью.

После выполнения этого кода в регистре АХ будет результат умножения чисел 150 и 250, то есть число 37500 (927С в [шестнадцатеричной системе](http://av-assembler.ru/asm/afd/asm-hex-system.htm)).

Теперь попробуем умножить 10000 на 5000.

MOV AX, 10000 ; Первый множитель в регистр AX

MOV BX, 5000 ; Второй множитель в регистр BX

**MUL** BX ; Теперь (DX АХ) = 10000 \* 5000 = 50000000

В результате мы получили довольно большое число, которое, конечно, не поместится в слово. Поэтому для результата используются два регистра - DX и AX. В нашем примере в регистре DX, будет число 762 (02FA - в шестнадцатеричной системе), а в регистре АХ - число 61568 (F080 - в [шестнадцатеричной системе](http://av-assembler.ru/asm/afd/asm-hex-system.htm)). А если рассматривать их как одно число (двойное слово), где в старшем слове 762, а в младшем - 61568, то это и будет 50000000 (2FAF080 - в шестнадцатеричной системе).

Если не верите - может перевести всё это в двоичное число и проверить.

Теперь о [флагах](http://av-assembler.ru/asm/afd/asm-flags-register.htm).

После выполнения команды MUL состояния флагов ZF, SF, PF, AF не определены и могут быть любыми.

А если старшая секция результата (регистр AH при умножении байтов или регистр DX при умножении слов) равна нулю, то

CF = OF = 0

Иначе эти флаги либо не равны, либо равны 1.

# Флаги

* CF - флаг переноса. Этот флаг может быть изменён, например, при переполнении.
* ZF - флаг нуля. Этот флаг будет установлен, если результатом сложения будет ноль
* SF - флаг знака. Этот флаг будет установлен, если результатом сложения будет отрицательное число.
* OF - флаг переполнения. Этот флаг будет установлен в случае переполнения (то есть когда результат не помещается в СУММУ - значение слишком большое).
* PF - флаг чётности. Этот флаг устанавливается в довольно сложных случаях (для начинающих это пока лишнее), поэтому подробно рассказывать не буду.
* AF - вспомогательный флаг переноса. Для новичков это тоже лишнее.

# Демонстрация работы программы

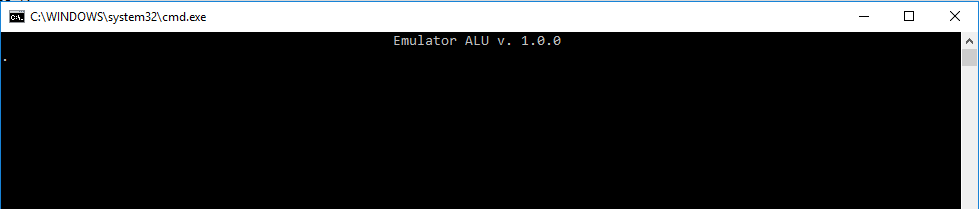


Рисунок 1 – Запуск программы

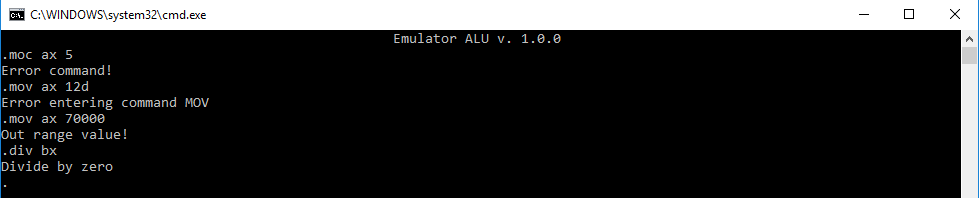


Рисунок 2 – Виды ошибок в программе

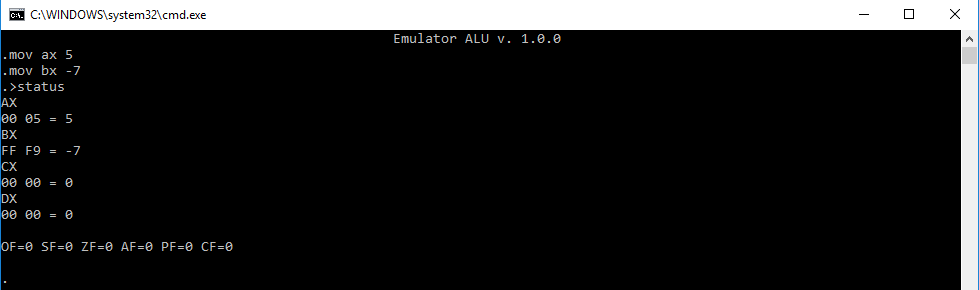


Рисунок 3 – Статус регистров и флагов

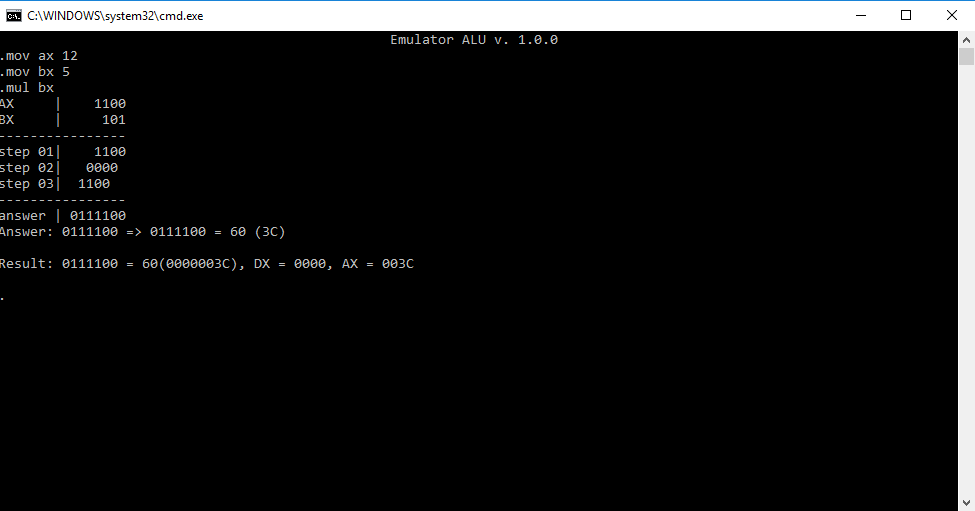


Рисунок 4 – Пошаговое выполнение операции умножения с положительным результатом

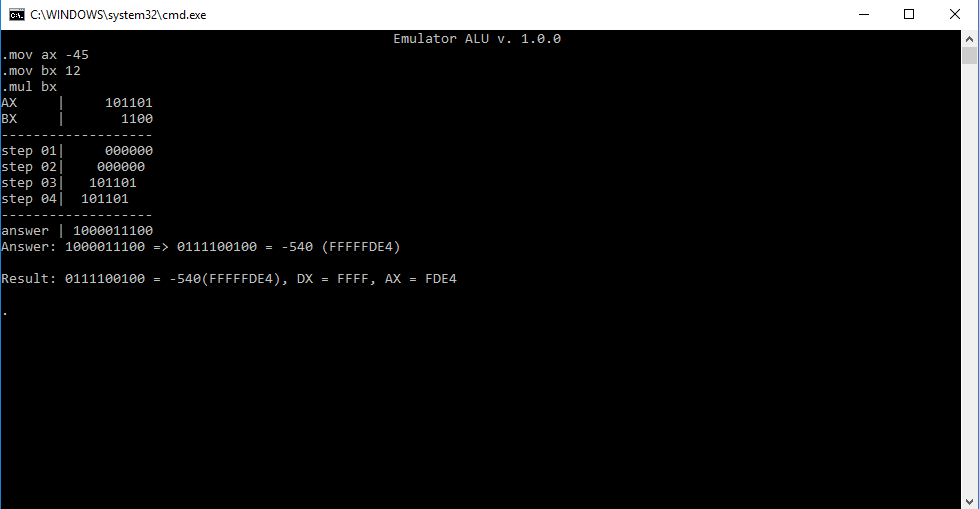


Рисунок 5 – Пошаговое выполнение операции умножения с отрицательным результатом

# Вывод

Достоинствами представления чисел в формате с *фиксированной запятой* являются простота и наглядность представления чисел, а также простота алгоритмов реализации арифметических операций.

Недостатком представления чисел в формате с *фиксированной запятой* является небольшой диапазон представления величин, недостаточный для решения математических, физических, экономических и других задач, в которых используются как очень малые, так и очень большие числа.

Во многих компьютерах умножение производится как последовательность сложений и сдвигов. Для этого в АЛУ имеется регистр, называемый накапливающим сумматором, который до начала выполнения операции содержит число ноль. В процессе выполнения операции в нем поочередно размещаются множимое и результаты промежуточных сложений, а по завершении операции — окончательный результат.

Другой регистр АЛУ, участвующий в выполнении этой операции, вначале содержит множитель. Затем по мере выполнения сложений содержащееся в нем число уменьшается, пока не достигнет нулевого значения.