# Практический раздел

## Указания по выбору варианта

Рабочей программой дисциплины «Архитектура компьютера» предусмотрено выполнение двух ИПР с ИКТ и двух контрольных работ.

Лабораторная работа должна быть представлена в виде исполняемых файлов и текстов программ. Отчет по лабораторной работе должен содержать задание, пример выполнения программы.

Контрольная работа должна быть оформлена в соответствии с общеустановленными нормами и правилами, предъявляемыми к выполнению контрольных работ.

Выбор вариантов осуществляется студентом самостоятельно на основании двух последних цифр номера зачетной книжки.

# ЛАБОРАТОРНЫЕ РАБОТЫ (ИПР С ИКТ)

# ПРОГРАММИРОВАНИЕ АРИФМЕТИЧЕСКОГО СОПРОЦЕССОРА

## ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

### *Сопроцессорные конфигурации*

Использование сопроцессора позволяет значительно ускорить работу программ, выполняющих расчеты с высокой точностью, тригонометрические вычисления и обработку информации, которая должна быть представлена в виде действительных чисел. Сопроцессор подключается к системной шине параллельно с центральным процессором (CPU) и может работать только совместно с ним. Все команды попадают в оба процессора, а выполняет каждый свои. Сопроцессор не имеет своей программы и не может осуществлять выборку команд и данных. Это делает центральный процессор. Сопроцессор перехватывает с шины данные и после этого реализует конкретные действия по выполнению команды. Два процессора работают параллельно, что повышает эффективность системы. Но возникают ситуации, когда их работа требует синхронизации (из-за разницы во времени выполнения команд).

**Синхронизация по командам**. Когда центральный процессор выбирает для выполнения команду FPU, последний может быть занят выполнением предыдущей команды. Поэтому перед каждой командой сопроцессора в программе должна стоять специальная команда (wait), которая только проверяет текущее состояние FPU и, если он занят, переводит центральный процессор в состояние ожидания. Соответствующую команду в программу может вводить либо ассемблер, либо компилятор языка без указаний программиста.

**Синхронизация по данным**. Если выполняемая в FPU команда записывает операнд в память перед последующей командой СРU, которая обращается к этой ячейке памяти, требуется команда проверки состояния FPU. Если данные еще не были записаны, СPU должен переходить в состояние ожидания. Автоматически учесть такие ситуации довольно сложно, поэтому вводить команды, которые проверяют состояние сопроцессора и при необходимости заставляют центральный процессор ожидать, должен программист.

### *Программная модель сопроцессора*

В программную модель любого процессора включаются только те регистры, которые доступны программисту на уровне машинных команд. Основу программной модели FPU образует регистровый стек из восьми 80-битных регистров R0-R7. В них хранятся числа в вещественном формате. В любой момент времени 3-битное поле ST в слове состояния определяет регистр, являющийся текущей вершиной стека и обозначаемый ST(0). При занесении в стек (push) осуществляется декремент поля ST и загружаются данные в новую вершину стека. При извлечении из стека (pop) в получатель, которым чаще всего является память, передается содержимое вершины стека, а затем инкрементируется поле ST .

В организации регистрового стека FPU есть отличия от классического стека.

1. Стек имеет кольцевую структуру. Контроль за использованием стека должен осуществлять программист. Максимальное число занесений без промежуточных извлечений равно 8.

2. В командах FPU допускается явное или неявное обращение к регистрам с модификацией или без поля ST. Например, команда fsqrt замещает число из вершины стека значением корня из него. В бинарных операциях допускается явное указание регистров. Адресация осуществляется относительно текущей вершины стека и обозначение ST (i) 0<i<7 , считая от вершины.

3. Не все стековые команды автоматически модифицируют указатель вершины стека.

С каждым регистром стека ассоциируется 2-битный тег (признак), совокупность которых образует слово тегов. Тег регистра R0 находиться в младших битах, R7 – в старших. Тег фиксирует наличие в регистре действительного числа (код 00), истинного нуля (код 01), ненормализованного или бесконечности (код 10) и отсутствие данных (код 11). Наличие тегов позволяет FPU быстрее обнаруживать особые случаи (попытка загрузить в непустой регистр, попытка извлечь из пустого) и обрабатывать данные.

Остальными регистрами FPU являются регистр управления, регистр состояния, два регистра состояния команды и два регистра указателя данных. Длина их всех 16 бит.

### *Форматы численных данных*

Арифметический FPU К1810ВМ87 оперирует с семью форматами численных данных, образующих три класса: двоичные целые, упакованные десятичные целые и вещественные числа. Во всех форматах старший (левый) бит отведен для знака.

Форматы различаются длиной, следовательно, диапазоном допустимых чисел, способом представления (упакованный и неупакованный формат), способом кодировки(прямой и дополнительный код).

**Двоичные целые числа.** Три формата целых двоичных (целое слово (16 бит), короткое целое (32 бита), длинное целое (64 бита)) отличаются длиной, следовательно, диапазоном чисел. Только в этих форматах применяется стандартный дополнительный код. 0 имеет единственное кодирование. Наибольшее положительное число кодируеться как 011…1, а Наибольшее по модулю отрицательное как 100..0.

**Упакованные десятичные целые.** Числа представлены в прямом коде и упакованном формате, т.е. в байте содержится две десятичные цифры в коде 8421. Старший бит левого байта – знак, остальные игнорируются, но при записи в них помещаются нули. Но надо учитывать, что при наличии в тетраде запрещающих комбинаций 1010 – 1111 результат операции не определен. Т.е. сопроцессор не контролирует правильность результата.

**Вещественные числа.** Различают короткие вещественные (КВ)(мантиса – 24 бита, порядок – 8 бит), длинные вещественные (ДВ) (мантиса – 53 бита, порядок – 11 бит) и временные вещественные (ВВ) (мантиса – 64 бита, порядок – 15 бит). Для них применяется формат с плавающей точкой. Значащие цифры находятся в поле мантисы, порядок показывает фактическое положение двоичной точки в разрядах мантисы, бит знака S определяет знак числа. Порядок дается в смещенной форме :

Е = истинный порядок + смещение

Смещение для соответствующих форматов равно 127, 1023, 16383 это упрощает операцию сравнения. Операция с целыми числами быстрее операции над плавающей точкой. Это важно в алгоритмах.

Значение числа равно

(1)S х 2E-смещение х F0F1F2…Fn,

где n для разных форматов равно 23,52 или 63.

Порядок имеет фиксированную длину, определяя один диапазон представимых чисел. Мантиса – правильная дробь. В коротком и длинном вещественном формате бит Fo при передаче чисел и хранении их в памяти не фигурирует. Это скрытый бит, который в нормализованных числах содержит 1. Скрытый бит не дает представить в этих форматах нуль и он должен кодироваться как спец значение.

Числа во временном вещественном формате имеют явный бит Fo. Формат повышает скорость выполнения операций благодаря простоте представления чисел, не являющихся ненормализованными. При загрузке из памяти в регистр FPU оно преобразуется во временный вещественный формат. А при записи в память – обратный формат. Временной вещественный формат – единственный, в котором абсолютно точно кодируется любые загружаемые из памяти числа.

### *Режимы работы. Состояние*

Сопроцессоримеет 2 доступных 16-битных регистра, содержимое которых определяет его режим работы и текущее состояние. Форматы регистров содержат слово управления CW и слово состояния SW. Регистр управления содержит 6 бит масок особых случаев. Регистр состояния – 6 бит флажков.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Регистр управления: | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| X | X | X | IC | RC | | PC | | IEM | X | PM | UM | OM | ZM | DM | IM |
| Регистр состояния: | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| B | C3 | ST | | | C2 | C1 | C0 | IR | X | PE | UE | OE | ZE | DE | IE |
| Рис 1. Форматы слова управления и слова состояния | | | | | | | | | | | | | | | |

Регистр управления содержит 6 бит масок особых случаев, а регистр состояния 6 бит флажков особых случаев:

P – потеря точности

U – антипереполнение

О – переполнение

Z - деление на нуль

D - денормализованный операнд

I - недествительный операция

**Слово управления.** Оно определяет для FPU один из нескольких вариантов обработки численных данных. Программа центрального процессора может сформировать в памяти образ слова управления, а затем заставить сопроцессор загрузить его в регистр CW. Рассмотрим значение полей.

Шесть младших бит слова управления - индивидуальные маски особых случаев. Т.е. особых ситуаций, обнаруженных FPU при выполнении команд. Если бит=1, то не будет вызвано прерывание СPU. Иначе FPU устанавливает в 1 бит запроса прерывания в слове состояния и при общем разрешении прерываний генерирует сигнал int прерывания СPU.

Бит 7 слова управления содержит маску управлени прерыванием IEM, которая разрешает (IEM=0) или запрещает(IEM=1) прерывание центрального процессора.

Двухбитное поле управления точностью (PC) определяет точность вычислений в 24 бита(РС=00), 53 бита (РС=10) или 64 бита (РС=11). По умолчанию вводиться режим с максимальной точностью в 64 бита.

Двухбитное поле управления округлением RC определяет один из четырех возможных вариантов округления результатов операций сопроцессора.

Бит 12 управляет режимом бесконечности IC. Когда IC=0, сопрцессор обрабатывает два специальных значения “плюс бесконечность” и “минус бесконечность” как одно и то же значение “бесконечность”, не имеющее знака.

**Слово состояния.** В нем младшие 6 бит отведены для регистрации особых случаев. Бит 7 – запроса прерывания (IR), устанавливается в 1 при возникновении любого незамаскированного особого случая. Бит С3-С0 фиксирует код условия в операциях сравнения, проверки условия и анализа. Три бита ST указатели стека. Стековые операции сопровождаются модификацией поля ST. Наконец, флажок занятости В устанавливается в состояние 1 когда численное операционные устройство выполняет операцию. Большую роль играют биты кода условия, которые фиксируют особенности результата (табл. 1.). Коды условия привлекаются для реализации условных переходов. Сопроцессор самостоятельно не может влиять на ход выполнения программ. Поэтому для условных переходов по результатам операций сопроцессора приходится сначала передавать код условия в память, а затем загружать один из регистров центрального процессора. После этого код условия передается в регистр флагов, производится условный переход.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 1. Интерпетация кода условия в операциях сравнения и проверки (FCOM,FCOMP,FTST) | | | | |
| C3 | C2 | C1 | C0 | Описание |
| 0 | X | X | 0 | (ST)>источника (src) |
| 0 | X | X | 1 | (ST)Бисточника (src) |
| 1 | X | X | 0 | (ST)=источнику (src) |
| 1 | X | X | 1 | Не сравнимы |

### *Система команд*

Система команд сопроцессора содержит 6 групп команд: команды передач данных, арифметические команды, команды сравнения, трансцендентных операций, команды загрузки констант и управления сопроцессором. Операнды некоторых команд определяются неявно. Другие команды допускают или требуют явного задания операнда.

При рассмотрении ситемы команд сопроцессора будем пользоваться следующими обозначениями: src обозначает источник, т.е. операнд, значение которого не модифицируеться, а dst – получатель, т.е. операнд, значение которого замещается результатом операции.

**Особенности задания команд.** Команды бинарных операций допускают несколько форм задания**.** При пустом поле операнда операция выполняется с двумя верхними элементами стека ST(источник) и ST(1)(получатель). После производства операции осуществляеться инкрементирование указателя стека и результат помещаеться в новую вершину стека, заменяя исходное содержимое ST(1). Когда в бинарной команде определен один операнд, операция выполняется с привлечением указанного в команде регистра или ячейки памяти и содержимого вершины стека. Результат загружается в старую вершину стека и указатель стека не модифицируется. Если в бинарной команде указаны 2 операнда, ими является содержимое 2-х регистров стека, причём одним из них будет ST, a 2-м ST(i).

Альтернативные формы операндов условно показываются с помощью наклонной черты, причем черта без последующей спецификации означает отсутствие явно задаваемых операндов. Например команда FADD имеет следующий общий вид:

FADD //src/dst,src

Такая запись подразумевает три возможных формы команд: без операндов, с одним источником, с получателем и источником.

В мнемониках команд сопроцессора приняты следующие соглашения: первая буква всегда F; вторая буква I обозначает операцию с целыми числами, буква B – операцию с десятичным целым операндом, а пустая – операцию с вещественными числами; предпоследняя или последняя буква R указывает обратную операцию (например, в обычной форме команды деления получатель делится на источник, а в обратной форме источник делится на получатель; в обоих формах результат помещаеться в получатель); последняя буква P идентифицирует команду, заключительным действием которой являеться извлечение из стека.

**Команды передачи данных.** Команды этой группы производят передачу данных между регистрами стека, а так же между вершинами стека и памятью. Одной командой число из памяти преобразуется во ременный вещественный формат и загружается в стек. Таким же образом, но в обратном порядке осуществляется передача числа в память.

**Команды загрузки.** 3 команды загрузки имеют следующий вид:

вещественное: FLD src

двоичное целое FILD src

десятичное целое FBLD src

Эти команды осуществляют декремент указания стека и передачу в новую вершину стека содержимого источника. В команде FLD источником может быть один из регистров стека или вещественное число. В командах FILD и FBLD – только операнд в памяти.

**Команды запоминания.**

Вещественное: FST dst

Двоичное целое: FIST dst

Они производят передачу содержимого вершины стека в память без модификации указателя ST и содержимого ST(0). В команде FST получатель регистр стека или вещественная переменная в памяти. В команде FIST получателем является переменная в памяти имеющая формат коротко целого и целого слова. Рассмотренные команды не допускают получателем формат длинного целого, временного вещественного и неупакованного десятичного.

**Команды запоминания с извлечением из стека.**  Три команды, помимо передачи содержимого ST(0) осуществляют извлечение из стека. Регистры бывшей вершины стека отмечается как пустой и производится инкремент указателя стека:

Вещественное: FSTР dst

Двоичное целое: FISTР dst

Десятичное целое: FВSTР dst

Действия команды FSTР очень похожи на действия FST с добавлением извлечения из стека. Однако, FSTР может передать в память слово во временном вещественном формате, чего не может сделать FST. Команда FIST обеспечивает передачу в память числа в любом формате целого двоичного, включая длинное целое. Последний формат не допустим в формате FSTР. Команда FВST преобразует операнд из вершины стека в упакованное десятичное число, передает его в память и производит извлечение из стека.

**Команда обмена содержимого регистров.**

FXCH//dst ST(0)<->(dst) обменивает содержимое получателя ST(i) с вершиной ST(0). При пустом поле операнда обменивается содержимое регистров ST(1) и ST(0).

**Команды управления.**

Команда FINIT / FNINIT - инициализировать сопроцессор.

Команда FLDCW src – загрузить слово управления. Источником является целое число в памяти.

Команды FSTCW dst и FSTSW dst – запомнить слово управления и состояния в ячейке памяти, определяемой получателем dst.

Команда FSTENV dst – запомнить среду. Под средой сопроцессора К1810ВМ87 понимаеся содержимое регистров управления, состояния, тегов, указателя команды и указателя операнда. Команда FSTENV передаёт его в область памяти с начальным адресом, указаным в команде. Формат хранения среды в памяти показан на рис. 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Начальный адрес → | 15 |  | 0 | Смещение |
| Слово управления | | | +0 |
| Слово состояния | | | +2 |
| Слово тегов | | | +4 |
|  | Указатель команды |  | +6 |
|  |  | +8 |
|  | Указатель операнда |  | +10 |
|  |  | +12 |
| Рис. 2. Формат хранения среды сопроцессора в памяти. | | | | |

До выборки из очереди следующей команды сопроцессора выполнение команды FSTENV должно закончиться.

Команда FLDENV src – загрузить среду. Парная предыдущей команде команда FLDENV src осуществляет загрузку среды сопроцессора из области памяти, определяемой srс. После команды FLDENV не требуется команда FWAIT, так как сопроцессор автоматически контролирует завершение передачи всех слов среды до перехода к своей следующей команде.

Команда FSAVE dst - сохранить полное состяние сопроцессора. Полное состояние сопроцессора представляет собой содержимое всех регистров програмной модели – среды и восьми регистров стека. Размер полного состояния сопроцессора составляет 94 байта. Команда FSAVE передаёт его в область памяти с начальным адресом, указаным в команде. Формат размещения полного состояния сопроцессора в памяти (его иногда называют «образом в памяти») показан на рис.3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Начальный адрес → | 15 |  | 0 | Смещение |
| Среда сопроцессора | | | +0 |
| Верхний элемент стека st(0) | | | +14 |
| Следующий элемент стека st(1) | | | +24 |
| ... | | | ... |
| Последний элемент стека st(7) | | | +12 |
| Рис. 3. Формат хранения полного состояния сопроцессора в памяти. | | | | |

Команда FRSTOR src – восстановить полное состояние сопроцессора.

Для сохранения и восстановления состояния сопроцессора обычно применяется следующий ассемблерный фрагмент:

SUB SP,94 ;Зарезервировать пространство в стеке

MOV BP,SP ;BP является базой для состояния

FSAVE [BP] ;Сохранить полное состояниеъ

…

MOV BP,SP ;BP является базой для состояния

FRSTOR [BP] ;Восстановить состояние

ADD SP,94 ;Освободить пространство в стеке

**Арифметические команды.**

Необходимо отметить, что вещественные числа в памяти, не могут быть в формате временного вещественного, а целые числа – в формате длинного целого. Здесь сказывается недостаточность наборов кодов операций.

**Команды сложения.** Операция сложения реализуется командами со следующими формами:

вещественные числа FADD //src/dst,src

вещественные числа с извлечением из стека FADDP dst,src

целые числа FIADD src

Отметим, что команда FADD ST,ST(0) удваивает содержимое вершины стека.

**Команды вычитания.** Обычное вычитание  осуществляют команды:

вещественные числа FSUB //src/dst,scr

вещественные числа с извлечением из стека FSUBP dst,src

целые числа FISUB src

Для производства обратного вычитания  предназначены команды FSUBR, FSUBRP, FISUBR, имеющие аналогичные формы.

**Команды умножение.** Операция умножение реализуется следующими командами:

вещественные числа FMUL //src/dst,scr

вещественные числа с извлечением из стека FMULP dst,src

целые числа FIMUL src

**Команды деления**. Для выполнения обычной операции деления предусмотрены команды:

вещественные числа FDIV //src/dst,scr

вещественные числа с извлечением из стека FDIVP dst,src

целые числа FIDIV src

Соответствующие команды обратного деления FDIVR, FDIVRP, FIDIVR загружают в получатель частное от деление источника на получатель.

Приведём несколько примеров арифметических команд:

FADD ST, ST(5) ; Сложить содержимое регистров

FIADD WORD PTR COUNT [SI] ; Прибавить целое слово

FSUBP ST(2), ST ; Вычесть содержимое регистров

FDIVR DWORD PTR [SI] ;Разделить короткое вещественное

FIDIVR DWORD PTR [BX+5] ; Разделить короткое целое

**Команда извлечения квадратного корня**. Команда FSQRT извлечения квадратного корня заменяет число, находящееся в вершине стека, значением квадратного корня:

FSQRT ST(0) <- ST(0)1/2.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Могут ли процессор и сопроцессор работать параллельно?

2. В чем суть синхронизации по данным?

3. В чем суть синхронизации по командам?

4. Какие регистры образуют программную модель сопроцессора?

5. C какими форматами численных данных может оперировать сопроцессор?

6. Что подразумевается под средой сопроцессора?

7. Что подразумевается под состоянием сопроцессора?

## ЛАБОРАТОРНЫЕ ЗАДАНИЯ

### *Задание 1.*

Написать программу, находящую решение квадратного уравнения

*ax2 + bx + c = 0*

с помощью сопроцессора.

### *Задание № 2.*

Значение аргумента **x** изменяется от ***a*** до ***b*** с шагом ***h***. Для каждого ***x*** найти значения функции Y(x), суммы S(x) и число итераций n, при котором достигается требуемая точность ε = |Y(x)-S(x)|. Результат вывести в виде таблицы. Значения ***a****,* ***b****,* ***h*** и ***ε*** вводятся с клавиатуры.

Работу программы проверить для *a*=0,1; *b*=0,8; *h*=0,1.

1. .

2. .

3. .

4. .

5. .

6. .

7. .

8. .

9. .

10. .

11. .

12. .

13. .

14. .

15..

16. .

17. .

18. .

19. .

20. .

**КОНТРОЛЬНЫЕ РАБОТЫ**

**Контрольная работа 1.** АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С ЦЕЛЫМИ ЧИСЛАМИ

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

### *Представление целых чисел.*

В двоичной системе счисления числа представляются с помощью комбинации единиц и нулей, знака "минус" и знака разделяющей точки между целой и дробной частью числа. Например, десятичное число -1.312510 в двоичном виде будет выглядеть как -1001.01012. Но в компьютере мы не можем хранить и обрабатывать символы знака и разделяющей точки — для "машинного" представления чисел могут использоваться только двоичные цифры (0 и 1). Если операции выполняются только с неотрицательными числами, то формат представления очевиден. В машинном слове из 8 бит можно представить числа в интервале от 0 до 255. Например:

|  |  |
| --- | --- |
| *00000000 =* | *0* |
| *00000000 =* | *0* |
| *00000001 =* | *1* |
| *00101001 =* | *41* |
| *10000000 =* | *128* |
| *11111111* | *255* |

В общем случае n-битовая последовательность двоичных цифр an-1an-2…a1a0 интерпретирована как целое число А, значение которого равно



#### *Прямой код*

Существует несколько соглашений о едином формате представления как положительных, так и отрицательных чисел. Всех их объединяет то, что старший бит слова (с точки зрения европейца — самый левый, или бит, которому при представлении числа без знака должен быть приписан самый большой вес) является битом хранения знака или знаковым разрядом. Все последующие биты слова представляют значащие разряды числа, которые в каждом формате интерпретируются по-своему. Значение 1 в знаковом разряде интерпретируется как представление всем словом отрицательного числа.

|  |  |
| --- | --- |
| 00010010 = | +18 |
| 10010010 = | -18 |

Общее правило математически формулируется следующим образом:

 (1)

Формат представления чисел в прямом коде неудобен для использования в вычислениях. Во-первых, сложение и вычитание положительных и отрицательных чисел выполняется по-разному, а потому требуется анализировать знаковые разряды операндов. Во-вторых, в прямом коде числу 0 соответствуют две кодовых комбинации:

|  |  |
| --- | --- |
| 00000000 = | +010 |
| 10000000 = | -010 |

Это также неудобно, поскольку усложняется анализ результата на равенство нулю, а такая операция в программах встречается очень часто.

Из-за этих недостатков прямой код практически не применяется при peaлизации в АЛУ арифметических операций над целыми числами. Вместо этого более широкое применение находит другой формат, получивший наименование дополнительного кода.

#### *Дополнительный код*

Как и в прямом, в дополнительном коде старший разряд в разрядной сетке отводится для представления знака числа. Остальные разряды интерпретируются не так, как в прямом коде. В табл. 1 перечислены основные свойства дополнительного кода и правила выполнения арифметических операций в дополнительном коде, которые мы рассмотрим в этом и следующем разделах.

Таблица 1. Свойства представления чисел в дополнительном коде

|  |  |
| --- | --- |
| *Диапазон представления на n-разрядной сетке* | от -2n-1 до 2n-1-1 |
| *Количество кодовых комбинаций, соответствующих числу 0* | Одна |
| *Отрицание* | Инвертировать значение в каждом разряде представления исходного числа (положительного или отрицательного), а затем сложить образовавшееся число с числом 0001 по правилам сложения чисел без знака |
| *Расширение разрядности* | Добавить дополнительные разряды слева и заполнить их представления значением, равным значению в знаковом разряде исходного представления |
| *Определение переполнения при сложении* | Если оба слагаемых имеют одинаковые знаки (оба положительны или оба отрицательны), то переполнение возникает в том и только в том случае, когда знак суммы оказывается отличным от знаков слагаемых |
| *Правило вычитания* | Для вычитания числа В из числа А инвертировать знак числа В, как описано выше, и сложить преобразованное число с А по правилам сложения в дополнительном коде |

В большинстве описаний дополнительного кода основное внимание уделяется технике формирования представления отрицательного числа по представлению соответствующего положительного, причем не приводится формальное доказательство работоспособности описанной схемы. Мы решили нарушить эту традицию, и в данном разделе, а также в следующем будем основываться на описании, в котором это представление рассматривается в терминах взвешенной суммы значений разрядов. Такой способ мы уже использовали выше при описании представления чисел без знака и целых чисел со знаком в прямом коде. Преимущество такой методики в том, что она не оставляет ни малейших сомнений в справедливости излагаемых правил выполнения арифметических операций в любых частных случаях.

Рассмотрим n-разрядное двоичное целое число А в дополнительном коде. Если А положительно, то значение его знакового разряда an-1 равно 0. В значащих разрядах будет представлена абсолютная величина числа точно так же, как и в прямом коде:



Число 0 считается положительным и, следовательно, в знаковом разряде его представления будет записан код 0, а во всех значащих разрядах также коды 0. Очевидно, что диапазон представления положительных чисел n-разрядным дополнительным кодом простирается от числа 0 до числа 2n-1-1 (для этого числа значения во всех значащих разрядах равны 1). Для представления большего числа потребуется расширение разрядной сетки.

Теперь перейдем к отрицательным числам. Знаковый разряд an-1 дополнительного кода отрицательного числа А (А<0) равен 1. В *n-*1 значащих разрядах может содержаться произвольная комбинация нулей и единиц, а таких комбинаций может быть 2n-1. Следовательно, имеется потенциальная возможность представить отрицательные числа от -1 до -2n-1 .Желательно таким образом установить соответствие между двоичными комбинациями и целыми отрицательными числами, чтобы арифметические операции над ним, выполнялись по тем же правилам, что и над числами без знака. В формате целых чисел без знака для вычисления значения числа по его двоичному представлению следует присвоить старшему разряду в разрядной сетке вес +2n-1. При представлении, включающем и знаковый разряд, это приводит к тому, что желаемые арифметические свойства сохраняются, если вес этого разряда будет равен -2n-1.Это соглашение используется при представлении чисел в дополнительном коде. Формально для отрицательного числа в дополнительном коде соблюдается соотношение

 (2)

Знаковый разряд an-1, дополнительного кода положительного числа равен 0 и, следовательно, член -2n-1an-1=0. Таким образом, соотношение (2) справедливо для дополнительного кода как положительных, так и отрицательных чисел.

В табл. 2 сравниваются представления 4-разрядных целых чисел в прямом и дополнительном кодах. Хотя с обычной точки зрения представление в дополнительном коде выглядит довольно экзотично, оно значительно упрощает правила выполнения арифметических операций сложения и вычитания. Поэтому такое представление используется для работы с целыми числами в подавляющем большинстве АЛУ современных процессоров.

Таблица 2. Варианты двоичного 4-разрядного представления целых чисел

|  |  |  |  |
| --- | --- | --- | --- |
| Десятичное  представление | Прямой код | Дополнительный код | Смещённое представление |
| +7 | *0111* | *0111* | *1111* |
| +6 | *0110* | *0110* | *1110* |
| +5 | *0101* | *0101* | *1101* |
| +4 | *0100* | *0100* | *1100* |
| +3 | *0011* | *0011* | *1011* |
| +2 | *0010* | *0010* | *1010* |
| +1 | *0001* | *0001* | *1001* |
| +0 | *0000* | *0000* | *1000* |
| -0 | *1000* | *-* | *0111* |
| -1 | *1001* | *1111* | *0110* |
| -2 | *1010* | *1110* | *0101* |
| *-3* | *1011* | *1101* | *0100* |
| *-4* | *1100* | *1100* | *0011* |
| *-5* | *1101* | *1011* | *0010* |
| *-6* | *1110* | *1010* | *0001* |
| *-7* | *1111* | *1001* | *0000* |
| *-8* | *-* | *1000* | *-* |

Хорошей иллюстрацией принципа представления в дополнительном коде является диаграмма веса разрядов (рис. 1), в которой показано, что вес самого младшего разряда (крайней правой позиции на диаграмме) равен 1 (т.е. 20 ). Вес каждого последующего — возрастает вдвое, и так до крайней левой позиции, знак веса которой инвертируется. Рис. 1,а дает представление о том, почему максимальное по абсолютной величине отрицательное число, которое можно представить в дополнительном коде, равно -2n-1. Код 1 в любом значащем разряде означает добавление во взвешенную сумму положительного числа, равного весу этого разряда. Очевидно также, что положительные числа должны иметь в знаковом разряде код 0, а отрицательные — код 1. Следовательно, самое большое положительное число должно иметь в знаковом разряде код 0, а во всех значащих — код 1 и будет равно 2n-1-1.

На рис. К.1 также показано, как можно использовать диаграмму веса разрядов для преобразования из десятичного представления в двоичное и наоборот.



#### *Преобразование при изменении длины разрядной сетки*

Иногда возникает необходимость записать n-разрядное целое двоичное число в слово длиной m бит, причем т>п. Если исходное число представлено в прямом коде, такое преобразование выполняется довольно просто — нужно перенести знаковый разряд в крайний левый бит нового слова, а остальные дополнительные биты заполнить нулями. Например:

|  |  |  |
| --- | --- | --- |
| *+18* | *00010010* | *прямой код, 8 разрядов* |
| *+ 18* | *00000000 00010010* | *прямой код, 16 разрядов* |
| *-18* | *10010010* | *прямой код, 8 разрядов* |
| *-18* | *10000000 00010010* | *прямой код, 16 разрядов* |

Но с отрицательными числами в дополнительном коде такая правильного результата. Рассмотрим тот же пример.

|  |  |  |
| --- | --- | --- |
| *+18* | *0010010* | *дополнительный код, 8 разрядов* |
| *+18* | *00000000 00010010* | *дополнительный код, 16 разрядов* |
| *-18* | *11101110* | *дополнительный код, 8 разрядов* |
| *32 658* | *10000000 01101110* | *дополнительный код, 16 разрядов* |

Почему в последней строке получился неправильный результат, станет понятно, если воспользоваться диаграммой веса (рис. 1) или формулой(2)

Преобразование дополнительного кода при расширении разрядной сетки выполняется следующим образом: нужно скопировать значение знакового бита во все дополнительные биты. Если исходное число было положительным, то все дополнительные биты заполнятся нулями, а если отрицательным – единицами. Эта операция называется расширением знака.Применив это правило к ранее рассмотренному примеру, получим:

|  |  |  |
| --- | --- | --- |
| *18* | *11101110* | *дополнительный код, 8 разрядов* |
| *-18* | *11111111 11101110* | *дополнительный код, 16 разрядов* |

Формально справедливость этого правила доказывается следующим образом. Рассмотрим n -разрядную последовательность двоичных цифр an-1аn-2...а1а0 которая интерпретируется как представление в дополнительном коде числа А:



Сразу видно, что если число А положительно, правило справедливо. А если число А отрицательно, нужно сформировать n-разрядное его представление (m>n), такое, что



Поскольку значения обоих представлений должны быть равны, то











Отсюда следует, что

*am-1 = am-2 = … = an = an-1 = 1*

В каждом из приведенных соотношений соблюдается условие неизменности младших n-1 разрядов представления. Последнее соотношение справедливо только в том случае, когда коды во всех разрядах от n-1 до т-2 равны 1. Тем самым подтверждается справедливость сформулированного выше правила расширения знака*.*

#### *Представление с фиксированной точкой*

И наконец, следует остановиться еще на одном нюансе. Описанные выше форматы объединяются часто одним термином — формат с фиксированной точкой. Суть его в том, что положение разделительной точки между целой и дробной частями числа неявно фиксируется на разрядной сетке. В настоящее время принято фиксировать точку справа от самого младшего значащего разряда. Программист может использовать аналогичное представление для работы с двоичными дробными числами, мысленно фиксируя точку перед старшим значащим разрядом и соответственно масштабируя результаты преобразований, выполняемых стандартными программными или аппаратными средствами.

### *Арифметические операций над целыми числами*

В этом разделе будут рассмотрены алгоритмы выполнения основных арифметических операций над целыми числами, представленными в дополнительном коде.

#### *Отрицание*

Операция отрицания числа, представленного в прямом коде, выполняется очень просто - нужно инвертировать значение знакового разряда. Если же число представлено в дополнительном коде, отрицание выполняется несколько сложнее. Правило выполнения этой операции формулируется следующим образом.

1. Следует инвертировать значение в каждом разряде представления исходного числа (положительного или отрицательного), включая и знаковый, т.е. установить значение 1 в тех разрядах, где ранее было значение 0, и значение 0 — в тех разрядах, где ранее было значение 1 (эту операцию иногда называют поразрядным дополнением — bitwise complement, а ее результат — инверсным кодом).
2. Нужно сложить образовавшееся число с числом 0. . .001 по правилам сложения чисел без знака.

Иногда эту операцию называют вычислением дополнения числа в дополнительном коде (twos complement operation). Например:

|  |  |  |
| --- | --- | --- |
| *+ 18=* | *00010010* | *(дополнительный код)* |
| поразрядное дополнение *=* | *11101101* |  |
| *+* | *1* |  |
|  | *11101110* | *=-18* |

Теперь проверим, будет ли выполняться правило, гласящее, что отрицание отрицания равно исходному числу:

|  |  |  |
| --- | --- | --- |
| *- 18=* | *11101110* | *(дополнительный код)* |
| поразрядное дополнение *=* | *00010001* |  |
| *+* | *1* |  |
|  | *00010010* | *=+18* |

Существуют два особых случая, на которых следует остановиться. Рассмотрим сначала случай А=0. Для восьмиразрядного представления такого числа получим

|  |  |  |
| --- | --- | --- |
| *0=* | *00000000* | *(дополнительный код)* |
| поразрядное дополнение *=* | *11111111* |  |
| *+* | *1* |  |
|  | ***1****00000000* | *=0* |

Выделенный в нижней строке код **1** есть не что иное, как перенос из старшего разряда, который игнорируется. Таким образом, результатом отрицания числа 0 в дополнительном коде будет дополнительный код числа 0, как и следовало ожидать.

Следующий особый случай порождает определенные проблемы. Речь идет о числе, которое в дополнительном коде имеет вид 100. .00, т.е. имеет в знаковом разряде код 1, а в п-1 значащих — код 0. После выполнения отрицания по описанному выше правилу получим тот же самый код:

|  |  |  |
| --- | --- | --- |
| *-128=* | *10000000* | *(дополнительный код)* |
| поразрядное дополнение *=* | *11111111* |  |
| *+* | *1* |  |
|  | *100000000* | *=-128* |

Подобные аномалии при работе с двоичными кодами неизбежны. В данном случае первопричина отклонения в следующем. Количество различных комбинаций п -разрядного двоичного кода равно 2п, т.е. является четным числомю С помощью этих комбинаций нам нужно представить положительные, отрицательные числа и число 0. Если представлять равное количество положительных и отрицательных чисел, то, как в прямом коде, получим две формы представления числа 0. Если же потребовать, чтобы числу 0 соответствовала только, единственная кодовая комбинация, то количества представляемых положительных и отрицательных чисел будут не равны. С помощью дополнительного кода можно представить число -2n, но нельзя представить число +2n .

#### *Сложение и вычитание в дополнительном коде*

Рассмотрим примеры:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *(а)* | *-7=* | *1001* |  | *(б)* | *-4=* | *1100* |  |
|  | *+5=* | *+0101* |  |  | *+4=* | *+0100* |  |
|  |  | *1110* | *=-2* |  |  | ***1****0000* | *=0* |
| *(в)* | *+3=* | *0011* |  | *(г)* | *-4=* | *1100* |  |
|  | *+4-* | *+0100* |  |  | *-1=* | *+1111* |  |
|  |  | *0111* | *=7* |  |  | *11011* | *=-5* |
| *(д)* | *+5=* | *0101* |  | *(е)* | *-7=* | *1001* |  |
|  | *+4=* | *+0100* |  |  | *-6=* | *+1010* |  |
|  |  | *1001* | *переполнение* |  |  | ***1****0011* | *переполнение* |

Первые четыре примера демонстрируют успешное выполнение операций. Если результат операции должен быть положительным, получается код положительного числа в дополнительном коде, а если отрицательным — код отрицательного числа в дополнительном коде. Обратите внимание на то, что в примере (г) формируется перенос из старшего (знакового) разряда, который игнорируется.

При выполнении сложения чисел с одинаковыми знаками результат может оказаться таким, что не вмещается в используемую разрядную сетку, т.е. получается число, которое выходит за диапазон представления. Появление такого результата расценивается как переполнение (overflow), и на схему АЛУ возлагается функция выявить переполнение и выработать сигнал, который должен воспрепятствовать использованию в дальнейшем полученного ошибочного результата. Существует следующее правило обнаружения переполнения:

Если знаки слагаемых совпадают, то переполнение возникает в том и только в том случае, когда знак суммы, полученной по правилам сложения в дополнительном коде, отличается от знака слагаемых.

Примеры (д) и (е) иллюстрируют появление переполнения при сложении положительных и отрицательных чисел. Обратите внимание на то, что переполнение может появиться и в том случае, когда возникает перенос из знакового разряда и когда перенос не возникает.

Операция вычитания выполняется по следующему правилу: Для вычитания одного числа (вычитаемого) из другого (уменьшаемого) необходимо предварительно выполнить операцию отрицания над вычит мым, а затем сложить результат с уменьшаемым по правилам сложен дополнительном коде.

Примеры выполнения с различными знаками.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *(а) М= 2 =* | *0010* | | | | *(б)* *М= 5 = 0101* | | | |
| *S= 7 =* | *0111* | | | | *S= 2 = 0010* | | | |
| *-S =* | *1001* | | | | *-S = 1110* | | | |
|  | *0010* | | | | *0101* | | | |
|  | *+ 1001* | | | | *+ 1110* | | | |
|  | *1011* | | | *=-5* | *3=* | | ***1****0011* | |
| *(в) М=-5 =* | *1011* | | | | *(г)* *М= 5 = 0101* | | | |
| *S=2 =* | *0010* | | | | *S=-2 = 1110* | | | |
| -S = | *1110* | | | | *-S = 0010* | | | |
|  | *1011* | | | | *0101* | | | |
|  | *+ 1110* | | | | *+ 0010* | | | |
|  | ***1****1001* | | *= -7* | | *7=* | | | *0111* |
| *(д) М= 7 =* | *0111* | | | | *(е)* *М=-6 = 1010* | | | |
| *S=-7 =* | *1001* | | | | *S= 4 = 0100* | | | |
| *-S =* | *0111* | | | | *-S= 1100* | | | |
|  | *0111* | | | | *1010* | | | |
|  | *+ 0111* | | | | *+ 1100* | | | |
|  | *1110* | *переполнение* | | | *переполнение* | ***1****0110* | | |

**

На рис. К.2 представлена блок-схема узлов АЛУ, принимавших участие в выполнении операций сложения и вычитания целых чисел .Центральным узлом является двоичный сумматор, на входы которого подают коды слагаемых, а на выходах формируется двоичный код суммы, причём реализация выполняется по правилам сложения чисел без знака. При выполнении сложения оба слагаемых направляются на входы сумматора непосредственно из peгистров слагаемых А и В.

Результат передается либо в один из регистров слагаемых (этот вариант показан на схеме), либо в третий регистр результата. Кроме кода результата сумматор формирует сигнал переполнения, который фиксируется в битовом флаге переполнения. Значение флага интерпретируется следующим образом: 0 — переполнение отсутствует, 1 – присуствует. При выполнении операции вычитания код вычитаемого, хранящийся перед началом операции в регистре В, передается на схему, выполняющую операцию отрицания, а уже с выхода этой схемы код поступает на вход сумматора.

#### *Умножение*

Алгоритмы выполнения умножения значительно сложнее, чем сложения или вычитания, причем в современных вычислительных системах можно встретить как аппаратную его реализацию, так и программную. Существует много вариантов этих алгоритмов, причем многие из них имеют не только теоретический, но и практический интерес, и выбор одного из многих может быть произведен только с учетом специфики применения конкретной cистемы. В данном разделе мы ставили перед собой задачу дать читателю общее представление о подходе, на основе которого такие алгоритмы npoeктируются. Начнем с простой задачи перемножения двух чисел без знака (т.е. неотрицательных чисел), а затем рассмотрим один из наиболее широко известных алгоритмов умножения целых чисел со знаком, представленных в двоичном коде.

Умножение чисел без знака

Ниже показана схема выполнения умножения двоичных чисел без знака, знакомая всем в десятичной интерпретации ещё со школьной скамьи, которую ещё называют умножением в столбик.

|  |  |
| --- | --- |
| *1011* | *Множимое(11)* |
| *х 1101* | *Множитель(13)* |
| *1011* |  |
| *0000* | *Частичные произведения* |
| *1011* |  |
| *1011* |  |
| *1000111* | *Произведение(143)* |

Анализируя этот пример, отметим следующее.

1. При выполнении умножения необходимо формировать частичные произведения, по одному на каждый разряд множителя. Эти частичные произведения затем суммируются, а их сумма и есть результат умножения — полное произведение.
2. Сформировать частичные произведения в двоичном коде довольно легко. Если соответствующий разряд множителя равен 0, частичное произведение  
   также равно 00..00. Если соответствующий разряд множителя равен 1  
   частичное произведение равно множимому.
3. Полное произведение вычисляется суммированием частичных произведений, причем каждое очередное частичное произведение в этой сумме сдвигается на одну позицию влево относительно предыдущего.
4. Результатом перемножения двух n-разрядных целых чисел будет 2n–разрядное число.

Реализация алгоритма умножения техническими или программными средствами позволяет несколько повысить его эффективность по сравнению с тем вариантом, который мы традиционно используем при вычислении в столбик вручную. Во-первых, суммирование очередного частичного произведения можно выполнять немедленно после того, как оно будет сформировано, не дожидаясь остальных. Таким образом, отпадает необходимость в средствах для временного хранения частичных произведений, т.е. для аппаратной реализации потребуется меньше регистров. Во-вторых, можно сберечь время, необходимое для формирования частичных произведений. Для каждого разряда в коде множителя, равного 1, необходимо выполнить сдвиг и сложение кода множимого, а для разряда, равного 0, — только сдвиг.

**

Возможный вариант реализации схемы умножения с учетом отмеченных усовершенствований алгоритма представлен на рис. 3. Множитель загружается в регистр Q, а множимое — в регистр М. В схеме имеется еще и третий регистр, регистр А, который в исходном состоянии сбрасывается в нуль. Одноразрядный регистр С, в который перед началом операции также записывается код 0, используется для хранения потенциального бита переноса, который может возникнуть в процессе суммирования.

Выполнение операции умножения в этой схеме происходит следующим образом. Схема управления сдвигом и сложением анализирует разряды множителе по одному, начиная с младших. Если Q0 равен 1, множимое складывается с содержимым регистра А и результат сохраняется в этом же регистре, причем регистр С используется для фиксации переполнения. Затем все разряды регистров С, А и Q сдвигаются на одну позицию вправо: содержимое С записывается в Аn-1, а0 переписывается в Qn-1 , а значение Q0 теряется. Если Q0 равен 0, сложение не выполняется, а только сдвигается содержимое регистров С, А и Q. Этот процесс выполняется для всех разрядов множителя. В результате в регистрах А и формируется 2n -разрядное произведение.

Ниже представлен пример выполнения умножения по такой схеме — изменение состояний регистров после каждого такта.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Исходные* *значения* | | *С 0* | *А 0000* | *Q*  *1101* | *М*  *1011* |
| *1-й такт* | *Сложение*  *Сдвиг* | *0 0* | *1101 0101* | *1101 1110* | *1011 1011* |
| *2-й такт* | *Сдвиг* | *0* | *0010* | *1111* | *1011* |
| *3-й такт* | *Сложение*  *Сдвиг* | *0 0* | *1101*  *0110* | *1111*  *1110* | *1011 1011* |
| *4-й такт* | *Сложение Сдвиг* | *0 0* | *0001 1000* | *1111*  *1111* | *1011* |

Умножение чисел в дополнительном коде

Мы уже отмечали, что при выполнении сложения и вычитания чисел в дополнительном коде они интерпретируются как числа без знака. Рассмотрим следующий пример:

|  |
| --- |
| *1101* |
| *+ 0011* |
| *1100* |

Если рассматривать операнды как числа без знака, то

*9(1001) + 3(0011) = 12(1100)*

А если интерпретировать их как числа в дополнительном коде *:*

*-7(1001) + 3(0011) = -4(1100)*

К сожалению, эта простая вычислительная схема оказывается неработоспособной при выполнении умножения. Чтобы убедиться в этом, вновь рассмотрим приведенный ранее пример. Мы умножали число 11 (1011) на 13 (1101) и в результате получили 143 (10001111). Но если интерпретировать сомножители как числа в дополнительном коде, то выходит, что умножение -5 (1011) на -3 (1101) дает -113 (10001111). Этот пример показывает, что такая простая схема неприменима, если оба сомножителя отрицательны. Фактически она не дает правильного результата в случае, если отрицателен хотя бы один из сомножит лей. Рассмотрим, как интерпретируется алгоритм умножения в столбик, если выразить его в терминах взвешенной суммы степеней числа 2. Тогда:

*1001 = 1х23+ 1х22+ 0x21 + 1x20= 23+22+20…*

Умножение двоичного числа на 2n реализуется сдвигом влево на п разрядов. Приняв это во внимание, получим следующую схему умножения рассматриваемых чисел:

|  |  |
| --- | --- |
| *1011*  *Х 1101* |  |
| *00001011* | *1011Х1Х20* |
| *00000000* | *1011Х0Х21* |
| *00101100* | *1011Х1Х22* |
| *01011000* | *1011Х1Х23* |
| *10001111* |  |

Единственное отличие этой схемы от представленной выше в том, что частичные произведения трактуются как 2n-разрядные числа, сформированные п-разрядного множимого.

Рассматривая исходное 4-разрядное множимое 1011 как число без знака, получим после расширения до восьми разрядов код 00001011. Любое частичное проиэведедение, соответствующее умножению этого числа на некоторый разряд множителя, отличный от 0-го (т.е. с весом 20), формируется сдвигом расширенного кода множимого влево на соответствующее число разрядов, причем освободившиеся справа разряды заполняются кодом 0. Например, в результате сдвигакода 00001011 на два разряда влево образуется код 00101100.

Теперь мы готовы показать, почему эта простая схема не работает в случае, когда множимое отрицательно. Дело в том, что каждое частичное произведение, образованное от отрицательного множимого, также является 2n-разрядным отрицательным числом и, следовательно, при его формировании знаковый разряд исходного числа должен быть расширен. В приведеном ниже примере показаны два варианта умножения двоичных кодов 1001 и 0011, причем в первом случае код 1001 интерпретируется как число без знака (его значение 9), а во втором — как число -7 в дополнительном коде. В первом случае получается результат 00011011, который соответствует в десятичном представлении числу 27 (и действительно, 9x3 = 27). Во втором же случае каждое частичное произведение формируется из расширенного (8-разрядного) кода 11111001 отрицательного множимого. Обратите внимание, что расширенный код можно сформировать, расширив влево значение знакового разряда исходного кода множимого.

|  |  |  |  |
| --- | --- | --- | --- |
| *Числа без знака* | | *Числа в дополнительном коде* | |
| *1001* | *(9)* | *1011* | *(-7)* |
| *х 0011* | *(3)* | *х 0011* | *(3)* |
| *00001001* | *1011х20* | *11111001* | *(-7)х20=(-7)* |
| *00010010* | *1011х21* | *11110010* | *(-7)х21=(-14)* |
| *00011011* | *(27)* | *11101011* | *(-21)* |

Теперь понятно, почему обычная схема умножения не будет работать и тогда, когда множитель является отрицательным числом. Причина в том, что разряды отрицательного множителя в дополнительном коде не будут при этом соответствовать частичным произведениям, сформированным сдвигом кода множимого. Например, 4-разрядный двоичный дополнительный код числа -3 имеет вид 1101. Если воспользоваться рассмотренной ранее схемой, то должна получиться последовательность сложения частичных произведений

*-(1х23+ 1х22+ 0x21+ 1x20) = -(23+22+20).*

А правильный результат должен иметь вид -(21+20). Следовательно, для управления суммированием частичных произведений нельзя напрямую исполь­зовать значения разрядов кода множителя так, как мы делали это ранее.

Решить эту дилемму можно по-разному. Один из способов состоит в том, чтобы преобразовать оба сомножителя в положительные числа, перемножить их по правилам умножения чисел без знака, а затем, если знаки сомножите­лей были разными, выполнить операцию отрицания результата по правилам, принятым для чисел в дополнительном коде. Однако конструкторы АЛУ предпочитают способ, который не требует выполнения дополнительного преобразования после завершения умножения. Одним из таких способов является алгоритм Бута (Booth). Этот алгоритм также позволяет ускорить выполнение операции. Схема алгоритма Бута приведена на рис. 5. Как и в ранее рассмотренном алгоритме, сомножители размещаются в регистрах Q (множитель) и М (множимое). Кроме них имеется одноразрядный регистр Q-1 который связан с младшим разрядом (Q0) регистра Q. Его назначение будет описано чуть ниже. Произведение формируется в регистрах А и Q. В исходном состоянии в регистрах А и Q , записаны нули. Как и ранее, схема управления анализирует разряды множителя, но на сей раз анализируется пара соседних разрядов — основной и тот, который находится справа от него. Если оба разряда имеют одинаковые значения (11 или 00), все разряды регистров A, Q и Q-1 сдвигаются на 1 разряд вправо. Если соседние разряды имеют отличающиеся коды, то выполняется сложение или вычитание кода множимого из содержимого регистра А. Сложение выполняется при комбинации кодов в соседних разрядах 01, а вычитание — при комбинации 10. Вслед за сложением или вычитанием выполняется сдвиг кодов в регистрах на один разряд вправо. Сдвиг выполняется таким образом, что старший разряд (крайний левый) регистра А (разряд Аn-1) сохраняется, хотя и переписывается в разряд Аn-2 . Это необходимо для сохранения знака кода в регистрах А и Q. Такую операцию принято называть сдвигом с сохранением знака или арифметическим сдвигом.



Ниже представлен пример выполнения умножения чисел 7 и 3 по такому алгоритму — изменение состояний регистров после каждого такта.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Исходные значения* | | *А*  *0000* | *Q*  *0011* | *Q-1*  *0* | *М*  *0111* |
| *1-й такт* | *А<-А-М*  *Сдвиг* | *1001*  *1100* | *0011*  *1001* | *0*  *1* | *0111*  *0111* |
| *2-й такт* | *Сдвиг* | *1110* | *0100* | *1* | *0111* |
| *3-й такт* | *А<-А+М*  *Сдвиг* | *0101*  *0010* | *0100*  *1010* | *1*  *0* | *0111*  *0111* |
| *4-й такт* | *Сдвиг* | *0001* | *0101* | *0* | *0111* |

Несколько примеров с разными значениями сомножителей показаны ниже. Как иллюстрируют эти примеры, алгоритм прекрасно справляется с перемножением сомножителей при любых комбинациях их знаков. Эффективность алгоритма повышается также за счет того, что при обнаружении в коде множителя последовательности из всех нулей или всех единиц операции сложения или вычитания не выполняются.

|  |  |  |  |
| --- | --- | --- | --- |
| *(а) 7x3=21* | | *(б) 7х(-3)=(-21)* | |
| *0111* |  | *0111* |  |
| *Х 0011* | *(0)* | *X 1101* | *(0)* |
| *11111001* | *10* | *11111001* | *10* |
| *0000000* | *11* | *0000111* | *01* |
| *000111* | *01* | *111001* | *10* |
| *00010101* | *(21)* | *11101011* | *(-21)* |
| *(а)(-7)х3=(-21)* | | *(б)7х(-3)=(-21)* | |
| *1001* |  | *1001* |  |
| *X 0011* | *(0)* | *Х 1101* | *(0)* |
| *00000111* | *10* | *00000111* | *10* |
| *0000000* | *11* | *1111001* | *01* |
| *111001* | *01* | *000111* | *10* |
| *11101011* | *(-21)* | *00010101* | *(21)* |

Теперь попробуем формально доказать, что алгоритм Бута дает правильный результат при любом сочетании знаков сомножителей. Сначала рассмотрим случай положительного множителя, в частности множителя, состоящего из непрерывной последовательности кодов 1, обрамленной кодами 0 (например, 000111110). Как известно, произведение есть сумма соответствующим образом сдвинутых копий множимого:

|  |  |
| --- | --- |
| *Mх(00011110)=* | *Mx(24+23+22+21)* |
| *=* | *Mx(16+8+4 + 2)* |
| *=* | *Mx30* |

Множество операций сложения можно заменить всего двумя, если принять во внимание, что

*2n +2n+1 + … + 2n-K = 2n+1 – 2n-K*

|  |  |
| --- | --- |
| *Mx(00011110) =* | *Mx(25-21)* |
| *=* | *Mx(32 - 2)* |
| *=* | *Мх30* |

Следовательно, произведение на такой множитель можно получить с помощью всего одной операции сложения и одной операции вычитания сдвинутого соответствующим образом множимого. Эту схему можно распространить на любое количество непрерывных последовательностей единиц в коде множимого и любую длину последовательности, включая и случай, когда последовательность состоит из одной единицы.

В алгоритме Бута используется именно эта схема: вычитание производится, когда встречается первая (крайняя справа) единица последовательности (сочетание разрядов Q0Q-1 равно 10), а сложение — когда встречается последняя (крайняя слева) единица последовательности (сочетание разрядов Q0Q-1 равно 01).

Эта же схема дает правильный результат и при отрицательном множителе.

Таким образом, работоспособность алгоритма Бута можно считать доказанной. По этому алгоритму выполняется вычитание множимого, когда обнаруживается первая справа единица в коде множителя (сочетание 10 в разрядах Q0Q-1), и сложение, когда последовательность единиц прерывается (сочетание 01 в разрядах Q0Q-1). Следующая операция вычитания множимого выполняется, когда обнаруживается начало очередной последовательности единиц.

Очевидно, что алгоритм Бута позволяет обойтись при выполнении умножения чисел в дополнительном коде меньшим количеством операций сложения и вычитания, чем простейший алгоритм.

#### *Деление*

По сравнению с умножением операция деления выполняется несколько сложнее, хотя соответствующие алгоритмы основываются на тех же принципах поразрядного анализа операндов. Исходный алгоритм, как и при умножении, — тот, который используется при вычислении вручную, карандашом на бумаге. Алгоритм состоит из повторяющейся последовательности шагов элементарных сдвигов и сложений или вычитаний.

На рис. К.6 показан пример "длинного" деления целых чисел без знака. Чрезвычайно полезно детально рассмотреть последовательность шагов этого алгоритма. Сначала разряды делимого анализируются слева направо до тех пор, пока последовательность разрядов не будет представлять число, большее, чем делитель, или равное ему. При делении этого числа на делитель получается частное, которое больше или равно 1. До тех пор, пока такое число не будет обнаружено при просмотре делимого(событие превышения), в частное заносятся коды 0, которые последовательно сдвигаются слева направо. Когда возникает событие превышения, в частное заносится 1, а делитель вычитается из частичного делимого. Результат этой операции называется частичным остатком. Далее выполняется циклическая процедура. В каждом цикле в частичный остаток добавляется очередной разряд делимого, и так до тех пор, пока при анализе частичного остатка не наступит событие превышения. В этом случае делитель вычитается из частичного остатка и формируется новый частичный остаток, а в очередной разряд частного заносится код 1. Процесс продолжается до тех пор, пока не будут проанализированы все разряды делимого.

На рис. К.7 представлена схема машинного алгоритма "длинного" деления. Перед началом выполнения операции делитель помещается в регистр М, делимое — в регистр Q, а регистр А очищается. На каждом шаге содержимое регистров А и Q сдвигается на 1 разряд влево. Содержимое регистра М вычитается из содержимого А, определяется, делится ли А на М (т.е. результат вычитания больше или меньше нуля). Если результат положительный, в младший разряд частного, Q0, заносится код 1. В противном случае в младший разряд частного вносится код 0, а значение частичного остатка (содержимое регистра А) восстанавливается, для чего к нему прибавляется содержимое регистра М. После этого уменьшается значение в счетчике циклов, в который перед началом выполнения операции записывается число n — количество элементарных циклов. После завершения n циклов частное будет находиться в регистре Q, а остаток — в регистре А.



**

Этот же алгоритм с небольшими изменениями можно использовать и для деления отрицательных чисел. Ниже мы опишем один из вариантов алгоритма деления чисел, представленных в дополнительном коде.

1. Загрузить делитель в регистр М, а делимое — в регистры А и Q. Делимое должно иметь формат 2n -разрядного дополнительного кода. Например, 4-разрядное число 0111 должно быть представлено в формате 00000111, а число 1001 должно быть преобразовано в 11111001.
2. Сдвинуть содержимое регистров А и Q на один разряд влево.
3. Если коды в регистрах М и А имеют одинаковые знаки, вычесть из содержимого А содержимое М и оставить результат в А. В противном случае добавить к содержимому А код из М.
4. Предыдущая операция считается успешной, если знак кода в регистре А не изменился в результате ее выполнения.
   * Если операция была успешной или содержимое регистров А и Q равно нулю, то установить в младшем разряде частного (разряде Q0) код 1.
   * Если операция не увенчалась успехом и содержимое одного из регистров А и Q (или обоих) отлично от нуля, то установить в младшем разряде частного, q0, код 0 и восстановить прежнее значение в регистре А.
5. Повторять операции, указанные в пунктах 2 и 4, столько раз, сколько разрядов в регистре Q.
6. После завершения операции прочесть значение остатка в регистре А. Если знаки делимого и делителя одинаковы, значение частного извлечь из регистра Q; в противном случае значение частного равно содержимому регистра Q с обратным знаком (операция отрицания должна быть выполнена по правилам для дополнительного кода).

Ниже приведены примеры выполнения деления по этому алгоритму при разных сочетаниях знаков операндов.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *(а) 7/3* | | | *(б) 7/(-3)* | | |  |  |
| *А* | *Q* | *М=0011* | *А* | *Q* | *M=1101* |  |  |
| *0000* | *0111* | *Исходные значения* | *0000* | *0111* | *Исходные значения* |  |  |
| *0000* | *1110* | *Сдвиг* | *0000* | *1110* | *Сдвиг* |  |  |
| *1101* |  | *Вычитание* | *1101* |  | *Сложение* |  |  |
| *0000* | *1110* | *Восстановление* | *0000* | *1110* | *Восстановление* |  |  |
| *0001* | *1100* | *Сдвиг* | *0001* | *1100* | *Сдвиг* |  |  |
| *1110* |  | *Вычитание* | *1110* |  | *Сложение* |  |  |
| *0001* | *1100* | *Восстановление* | *0001* | *1100* | *Восстановление* |  |  |
| *ООН* | *1000* | *Сдвиг* | *0011* | *1000* | *Сдвиг* |  |  |
| *0000* |  | *Вычитание* | *0000* |  | *Сложение* |  |  |
| *0000* | *1001* | *Q0=l* | *0000* | *1001* | *Q0=l* |  |  |
| *0001* | *0010* | *Сдвиг* | *0001* | *0010* | *Сдвиг* |  |  |
| *1110* |  | *Вычитание* | *1110* |  | *Сложение* |  |  |
| *0001* | *0010* | *Восстановление* | *0001* | *0010* | *Восстановление* |  |  |
| *(в) (-7)/3* | | | *(г) (-7) / (-3)* | | |  |  |
| *А* | *Q* | *М=0011* | *А* | *Q* | *М=1101* |  |  |
| *1111* | *1001* | *Исходные значения* | *1111* | *1001* | *Исходные значения* |  |  |
| *1111* | *0010* | *Сдвиг* | *1111* | *0010* | *Сдвиг* |  |  |
| *0010* |  | *Сложение* | *0010* |  | *Вычитание* |  |  |
| *1111* | *0010* | *Восстановление* | *1111* | *0010* | *Восстановление* |  |  |
| *1110* | *0100* | *Сдвиг* | *1110* | *0100* | *Сдвиг* |  |  |
| *0001* |  | *Сложение* | *0001* |  | *Вычитание* |  |  |
| *1110* | *0100* | *Восстановление* | *1110* | *0100* | *Восстановление* |  |  |
| *1100* | *1000* | *Сдвиг* | *1100* | *1000* | *Сдвиг* |  |  |
| *1111* |  | *Сложение* | *1111* |  | *Вычитание* |  |  |
| *1111* | *1001* | *Q0=l* | *1111* | *1001* | *Q0=l* |  |  |
| *1111* | *0010* | *Сдвиг* | *1111* | *0010* | *Сдвиг* |  |  |
| *0010* |  | *Сложение* | *0010* |  | *Вычитание* |  |  |
| *1111* | *0010* | *Восстановление* | *1111* | *0010* | *Восстановление* |  |  |

В приведенных выше примерах получились разные значения остатка при делении пар чисел (-7)/3 и 7/(-3). Это произошло потому, что остаток определяется по формуле:

D = QxV + R,

где D — делимое, Q — частное, V — делитель, a R остаток. Приведенные выше примеры вполне согласуются с этой формулой.

**Порядок выполнения работы**

Изучить теоретический материал и выполнить задания 1 в соответствии со своим вариантом

### *Задание 1.*

Написать по шагам последовательность действий, реализующих *Операции сложения, вычитания, умножения и деления с фиксированной точкой*  над двумя числами.

**Контрольная работа 2.** АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С С ЧИСЛАМИ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

### *Представление чисел в формате с плавающей точкой.*

#### *Основные положения*

В формате с фиксированной точкой, в частности в дополнительном коде, можно представлять положительные и отрицательные числа в диапазоне, симметричном на числовой оси относительно точки 0. Расположив воображаемую Разделяющую точку в середине разрядной сетки, можно в этом формате представлять не только целые, но и смешанные числа, а также дроби.

Однако такой подход позволяет представить на ограниченной разрядной сетке множество вещественных чисел в довольно узком диапазоне. Нельзя представить очень большие числа или очень маленькие. При выполнении деления двух больших чисел, как правило, теряется дробная часть частного.

При работе в десятичной системе счисления ученые давно нашли выход из положения, применяя для представления числовых величин так называемую научную нотацию. Так, число 976 000000 000 000 можно представить в виде 9.76x1014, а число 0,000000 000 000 0976 - в виде 9.76x10-14. При этом, фактически, разделительная точка динамически сдвигается в удобное место, а для того чтобы "уследить" за ее положением в качестве второго множителя - характеристики, - используется степень числа 10 (основания характеристики). Это позволяет с помощью небольшого числа цифр (т.е. чисел с ограниченной разрядностью) с успехом представлять как очень большие, так и очень малые величины.

Этот же подход можно применить и в двоичной системе счисления. Число можно представить в виде

*±SxB±E*

Компоненты такого представления можно сохранить в двоичном слове, с, стоящем из трех полей:

* поле знака числа (плюс или минус)
* поле мантиссы S;
* поле порядка E;

Основание В подразумевается неявно и не сохраняется.

Принципы представления лучше пояснить на примерах. На рис. 8 схематически показан 32-х разрядный формат с плавающей точкой. В крайнем левом бите слова хранится знак числа(0 – положительное, 1- отрицательное). В следующих 8 битах хранится значение порядка. Для представления порядка используется так называемый смещенный формат. Для получения действительного двоичного кода порядка из значения, сохраняемого в этом поле нужно вычесть фиксированное смещение. Как правило, смещение равно (2k-1-1), где k – разрядность поля порядка.. В данном случае k = 8, и в поле порядка можно представить коды в диапазоне от 0 до 255. Если принять значение смещения 127, то действительное значение порядка чисел, представленных в таком формате может находится в интервале от -127 до +128. В данном примере считается, что основание характеристики совпадает с основанием системы счисления и равно 2.

Выше в табл. 2 были представлены смещенные представления 4-разрядных целых чисел. Заметьте, что если интерпретировать смещенное ставление как обычный формат без знака, то отношение старшинства между кодами сохраняется, т.е. наибольшее значение в обоих случаях есть 1111,а наименьшее — 0000. Этого не наблюдается при представлении чисел в прямом или дополнительном кодах. Преимущество смещенного представления порядка в формате числа с плавающей точкой в том, что результат сравнения двух неотрицательных вещественных чисел будет таким же, как и результат сравнения их кодов, рассматриваемых как целые числа без знака.

Последнее поле в слове (23 бит) отводится для хранения значения мантиссы S. Теперь хочу обратить ваше внимание на следующий нюанс. Любое число можно представить в форме с плавающей точкой множеством способов. Так, приведенные ниже формы представления эквивалентны, если считать, что мантисса выражена в двоичной системе счисления:

*0.110х25*

*110х22*

*0.0110х26*

Для упрощения алгоритмов выполнения арифметических операций обычно принято нормализовать мантиссу. Нормализованная мантисса числа, отличного от нуля, имеет вид

0.1bbb...bх2±Е,

где b представляет произвольную двоичную цифру (0 или 1). Это означает, что старший (левый) значащий разряд кода мантиссы всегда равен 1. Но если он всегда равен 1, его нет смысла хранить в составе числа, а можно просто учитывать этот факт при выполнении операций. Таким образом, в 23-битовом поле фактически хранится 24-разрядный код мантиссы, значение которой может был в диапазоне от 0.510 до 1.0.



Ниже приведены примеры чисел в формате с плавающей точкой.

|  |  |
| --- | --- |
| *0.11010001х210100 =* | *0 10010011 10100010000000000000000* |
| *-0.11010001х210100 =* | *1 10010011 10100010000000000000000* |
| *0.11010001х2-10100 =* | *0 01101011 10100010000000000000000* |
| *-0.11010001х2-10100 =* | *1 01101011 10100010000000000000000* |

Обратите внимание на следующие особенности:

* знак сохраняется в старшем бите слова;
* первый разряд мантиссы всегда равен 1, и в поле мантиссы не хранится;
* к действительному значению порядка прибавляется смещение 127 и в поле порядка хранится эта сумма;
* основание характеристики равно 2.

Если в слове такой же длины хранить целые числа в дополнительном коде, то диапазон представления будет охватывать 232 чисел от 2-31 до 231-1 включительно. В формате с плавающей точкой с распределением полей, как показано на рис. 8.10, можно хранить:

отрицательные числа от -(1-2-24)х2128 до -0.5х2-127,

положительные числа от 0.5х2-127 до (1-2-24)х2128.

Пять областей на числовой оси не включены в диапазон представления:

* отрицательные числа, меньшие, чем -(1-2-24)х2128, эта область именуется отрицательной областью переполнения (negative overflow);
* отрицательные числа, большие, чем -0.5х2-127, эта область именуется отрицательной областью потери значимости (negative underflow);
* нуль;
* положительные числа, меньшие, чем 0.5х2-127, эта область именуется положительной областью потери значимости (positive underflow);
* положительные числа, большие, чем (1-224)х2128, эта область именуется положительной областью переполнения (positive overflow).

Строго говоря, описанное выше представление не позволяет хранить число 0, но на практике код, состоящий из нулей во всех разрядах, считается допустимымипредставляет число 0. В области переполнения можно попасть в том случае, если результат арифметической операции имеет абсолютную величину, превышающую ту, которая представляется нормализованным числом с порядком 128, а в область потери значимости – если результат операции имеет очень маленькую абсолютную величину. Потеря значимости как правило не представляет особой проблемы, поскольку в таких случаях с достаточной точностью результат можно считать равным 0.

Обращаю ваше внимание на то, что формат с плавающей точкой не позволяет представить больше отличающихся друг от друга числовых величин. Их по-прежнему 232 (для слова в 32 бита). Интервал между соседними числами – переменный и зависит от абсолютной величины числа.. Фактически результаты округляются с точностью, определяемой имеющейся разрядной сеткой.

В рассматриваемом примере формата под порядок отводится 8 бит, а под мантиссу — 23 бит. Если увеличить длину поля порядка, диапазон представления расширится, но при этом придется сократить поле мантиссы, а значит точность представления снизится, т.е. уменьшится "плотность" размещения представляемых величин на числовой оси. Единственный способ увеличить диапазон, и точность представления — расширить разрядную сетку. В большинстве компьютеров имеются два формата с плавающей точкой: обычный и удвоенной точности. Например, типовой формат обычной точности занимает 32-битовое слово, а формат удвоенной точности занимает 64 бит.

#### *Стандарт IEEE формата с плавающей точкой*

Для унификации формата представления чисел с плавающей точкой, что явля­ется необходимым условием переносимости программного обеспечения, Институтом инженеров по электротехнике и радиоэлектронике IEEE разработан стандарт 754 . В последнее десятилетие практически все процессоры и арифметические сопроцессоры проектируются с учетом требований этого стандарта.

Стандарт специфицирует два варианта формата: 32-битовый — обычной точности представления и 64-битовый — удвоенной точности представления. В первом формате поле порядка занимает 8 бит, а во втором -11 бит. Стандарт регламентирует использование числа 2 в качестве неявно заданного значения основания характеристики. Помимо основных, в стандарте предусмотрены два расширенных варианта форматов обычной и удвоенной точности, конкретная спецификация которых зависит от реализации вычислительной системы. Расширенные форматы позволяют включать дополнительные биты в поле порядка (расширение диапазона представления) в поле мантиссы (повышение точности представления). Расширенные форматы предназначаются для промежуточных вычислений. За счёт повышения точности снижается вероятность появления ошибок округления, а при расширении диапазона снижается вероятность появления ошибки переполнения.

Таблица 3. Параметры форматов, регламентированные стандартом IEEE 754

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Параметр* | *Формат* | | | |
|  | *Обычная точность* | *Расширенный обычной точности* | *Удвоенная точность* | *Расширенный удвоенной точности* |
| *Размер слова (бит)* | *32* | *≥43* | *64* | *≥79* |
| *Поле порядка (бит)* | *8* | *≥11* | *11* | *≥15* |
| *Смещение порядка* | *127* | *Не регламентируется* | *1023* |  |
|  |
| *Максимальное значение порядка* | *127* | *≥1023* | *1023* | *≥16383* |
| *Минимальное значение* *порядка* | *-126* | *≤-1022* | *-1022* | *≤-16382* |
|  |  |  |  |
| *Диапазон представления* *(по основанию 10)* | *10-38, 10+38* | *Не регламентируется* | *10-308, 10+308* | *Не регламентируется* |
| *Поле мантиссы (бит)* | *23* | *≥31* | *52* | *≥63* |
| *Количество значений* *порядка* | *254* | *Не регламентируется* | *2046* | *Не регламентируется* |
|  |  |
| *Количество значений* *мантиссы* | *223* | *Не регламентируется* | *252* | *Не регламентируется* |
|  |  |
| *Количество отличающихся представимых* *величин* | *1.98х231* | *Не регламентируется* | *1.99х263* | *Не регламентируется* |
|  |  |
|  |  |  |  |

Не включая неявный старший разряд, равный 1

Не все двоичные комбинации в форматах, регламентированных стандартом, интерпретируются обычным способом (табл. 4). Крайние значения порядка — все нули (значение 0) и все единицы (значение 255 в формате обычной точности и 2047 в формате удвоенной точности) — представляют особые величины. Стандартом предусматривается представление в заданном формате следующих классов числовых величин.

* При значениях кода в поле порядка в интервале от 1 до 254 в формате обычной точности и от 1 до 2046 в формате удвоенной точности представляются нормализованные вещественные числа. Используется смещенный формат представления порядка и, следовательно, интервал возможных значений порядка простирается от -126 до +127 в формате обычной точности и от -1022 до +1023 в формате удвоенной точности. Нормализованная мантисса включает бит, равный 1, слева от разделительной точки, который не хранится в поле мантиссы. Следовательно, в формате обычной точности представляется 24-разрядная мантисса, а в формате удвоенной точности — 53-разрядная мантисса (в стандарте используется термин fraction — дробная часть).
* Нулевое значение в поле порядка совместно с нулевым значением в поле мантиссы представляет значение +0 или -0 в зависимости от кода в бите знака мантиссы.
* Заполнение поля порядка кодом 1 совместно с нулевым кодом в поле мантиссы представляет величины +∞ и -∞ в зависимости от кода в бите знака мантиссы. Таким образом, формат позволяет представить на ограниченной разрядной сетке неограниченно большую числовую величину. Пользователь (конструктор вычислительной системы) должен решить, трактовать ли переполнение как ошибку или обрабатывать в программе бесконечно большие числовые величины.
* Нулевое значение в поле порядка совместно с ненулевым значением в поле мантиссы представляет ненормализованные числа. В этом случае считается, что неявный старший разряд мантиссы равен 0, и значение порядка равно -126 в формате обычной точности или -1022 — в формате удвоенной точности. Код в знаковом бите мантиссы по-прежнему определяет знак числа.
* Заполнение поля порядка кодом 1 совместно с ненулевым кодом в поле мантиссы представляет так называемые необычные числовые типа NaN - Not a Number).

Таблица 4. Представление величин в формате с плавающей точкой в соответствии со стандартом IEEE 754

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *Обычная точность (32 бит)* | | | | *Удвоенная точность (64 бит)* | | | |
|  | *Знак мантисс­­сы* | *Смеще­нный*  *порядок* | *Ман­ти­сса* | *Значе­ние* | *Знак мантис­сы* | *Смеще­н­ный порядок* | *Мантис­са* | *Значе­ние* |
| Положитель­ный нуль | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* |
| Отрицательный  нуль | *1* | *0* | *0* | *-0* | *1* | *0* | *0* | *-0* |
| Плюс  бесконечность | *0* | *255 (все единицы)* | *0* | *+ ∞* | *0* | *2047 (все единицы)* | *0* | *+ ∞* |
| Минус бесконечность | *1* | *255 (все единицы)* | *0* | *-∞* | *1* | *2047 (все единицы)* | *0* | *-∞* |
| Простое значение  типа NaN | *0 или 1* | *255 (все*  *единицы)* | *≠0* | *NaN* | *0 или 1* | *2047 (все единицы)* | *≠0* | *NaN* |
| Сигнализирующее значение типа NaN | *0 или 1* | *255 (все единицы)* | *≠0* | *NaN* | *0 или 1* | *2047 (все единицы)* | *≠0* | *NaN* |
| Положительное нормализованное число,  отличное от 0 | *0* | *0<е<255* | *f* | *2e-127(1.f)* | *0* | *0<е<2047* | *f* | *2e-1023(1.f)* |
| Отрицательное нормализованное число,  отличное от 0 | *1* | *0<е<255* | *f* | *-2e-127(1.f)* | *1* | *0<е<2047* | *f* | *-2e-1023(1.f)* |
| Положительное ненормализованное число | *0* | *0* | *f≠0* | *2e-126(0.f)* | *0* | *0* | *f≠0* | *2e-1022(0.f)* |
| Отрицательное ненормализован­ное число | *1* | *0* | *f≠0* | *-2e-126(0.f)* | *1* | *0* | *f≠0* | *-2e-1022(0.f)* |

### *Арифметические операции над числами в формате с плавающей точкой.*

Таблица.5 Арифметические операции над числами в формате с плавающей точкой.

|  |  |
| --- | --- |
| *Представление чисел* | *Арифметические операции* |
|  |  |
|  |
|  |  |
|  |  |

*Примеры:*

*X = 0.3x102 = 30;*

*Y = 0.2x103 = 200;*

*X + Y = (0.3x102-3 + 0.2)x103 = 0.23x103 = 230;*

*X - Y = (0.3x102-3 - 0.2)x103 = (-0.17)x103 = -170;*

*X x Y= (0.3x0.2)x102+3 = 0.06x105 = 6000;*

*X / Y = (0.3/0.2)x102-3 = 1.5x10-1 = 0.15*

При сложении и вычитании необходимо предварительно уравнять порядки операндов, что требует сдвига положения разделительной точки в мантиссе. Операции умножения и деления уравнивания порядков операндов не требуют.

При выполнении операций могут возникнуть следующие особые ситуации.

* **Переполнение порядка.** Положительный порядок результата превышает максимальное значение, предусмотренное форматом. В некоторых системах такой результат может трактоваться как величина +∞ или -∞.
* **Потеря значимости порядка.** Отрицательный порядок результата меньше минимального значения, допускаемого принятым форматом (например, получен порядок -200, который меньше разрешенного -127). Это означает, что получен очень малый результат, который можно считать равным нулю.
* **Потеря значимости мантиссы.** В процессе уравнивания порядков мантисса сдвигается настолько сильно вправо, что старший значащий ее разряд выходит за пределы разрядной сетки.
* **Переполнение мантиссы.** При сложении мантисс с одинаковыми знаками возможно появление переноса из старшего разряда

#### *Сложение и вычитание*

Алгоритмы выполнения операций сложения и вычитания в формате с плавающей точкой сложнее, чем аналогичные алгоритмы для чисел в формате с фиксированной точкой. Связано это, в первую очередь, с необходимостью выравнивания порядков операндов. Алгоритм включает четыре основных этапа.

1. Проверка на нуль.
2. Сдвиг мантисс для выравнивания порядков.
3. Суммирование или вычитание мантисс.
4. Нормализация результата.

Блок-схема типового алгоритма представлена на рис. 9. Детальный пошаговый анализ этого алгоритма покажет, какие функции используются при выполнении операций сложения и вычитания чисел в формате с плавающей точкой. В дальнейшем для определенности будем считать, что используется формат, регламентированный стандартом IEEE 754. Перед началом выполнения операций операнды должны быть помещены в регистры АЛУ. Если в используемом формате с плавающей точкой предполагается неявный старший разряд мантиссы, этот разряд должен быть в явном виде включен в регистры операндов, и все операции с ним в дальнейшем будут проводиться точно так же, как и с остальными разрядами мантиссы.

Поскольку операции сложения и вычитания отличаются только тем, что при вычитании предварительно изменяется знак второго операнда (вычитаемого), эта операция включена в ветвь ВЫЧИТАНИЕ на схеме алгоритма, после чего обе ветви сливаются. Далее анализируется, не равен ли один из операндов нулю. Если это так, то результат — значение второго операнда.

Следующий этап — изменение кодов операндов таким образом, чтобы значения их порядков стали равны. Чтобы понять, зачем это нужно, рассмотрим следующий пример в десятичной системе счисления:

*(123х100) + (456х10-2).*

Очевидно, что нельзя просто сложить мантиссы этих двух чисел. Сначала нужно выровнять разрядные сетки обеих мантисс так, чтобы соответственные разряды (разряды с равным весом) занимали одинаковые позиции, т.е. цифра 4 второго числа находилась в той же позиции, что и цифра 3 первого числа. При этом порядки обоих чисел будут равны. Равенство порядков и есть, с точки зрения математики, условие, позволяющее складывать мантиссы обоих чисел в такой форме представления. Следовательно,

*(123х100) + (456х10-2) - (123х100) + (4.56х100) = 127.56х100.*

Выравнивание выполняется за счет сдвига мантиссы меньшего числа вправо или мантиссы большего числа — влево. Поскольку в любом варианте теряются цифры операнда, выполняется сдвиг мантиссы меньшего числа вправо, что приводит к утере ее младших разрядов. Одновременно со сдвигом мантиссы вправо порядок меньшего числа увеличивается. Сдвиги выполняются до тех пор пока значения порядков обоих чисел не станут равны.

После того, как порядки будут выровнены, наступает этап сложения мантисс с учетом их знаков. Поскольку слагаемые могут иметь разные знаки, их алгебраическая сумма может оказаться равной нулю. Не исключено и появление переполнения — переноса из старшего разряда суммы. В этом случае необходимо выполнить дополнительный сдвиг результата вправо и одновременно увеличить на 1 значение порядка. Но тогда возможно появление переполнения порядка, которое расценивается как аварийная ситуация и влечет за собой прекращение операции.

После сложения мантисс наступает этап нормализации результата. Нормализация представляет собой серию сдвигов кода мантиссы влево (в сторону старших разрядов) с одновременным уменьшением значения порядка до тех пор, пока значение старшей цифры мантиссы не станет отличным от нуля. Я специально оговариваю цифры, поскольку в случае, когда основанием характеристики является число 16, шестнадцатеричной цифрой мантиссы будет служить 4-разрядный двоичный код. Последняя операция – округление результата.



#### *Умножение и деление*

При работе с числами в формате с плавающей точкой алгоритмы умножения и деления оказываются проще алгоритмов сложения и вычитания.

**

Сначала рассмотрим алгоритм умножения (рис. К.10). Сразу же посче начала операции проверяется, не равен ли нулю один из сомножителей Если это так, то произведение также будет равно нулю. Следующий шаг — суммирование порядков. Поскольку, как правило, для хранения порядков используется смещенное представление, при суммировании двух смещенных представлений результат будет смещен дважды. Поэтому после суммирования кодов порядков из суммы вычитается значение смещения. При суммировании может возникнуть как переполнение порядка, так и потеря значимости. В обоих случаях формируется соответствующий сигнал. Если порядок произведения не выходит из диапазона, определенного форматом, далее перемножаются мантиссы сомножителей с учетом их знаков. Умножение мантисс выполняется по тому же алгоритму, что и умножение целых чисел в прямом коде, т.е. фактически перемножаются числа без знака, а затем произведению приписывается знак "плюс" или "минус" в зависимости от сочетания знаков сомножителей. Произведение мантисс имеет разрядность, вдвое большую, чем каждый из сомножителей. Лишние младшие разряды отбрасываются при округлении.

После того как будет получено произведение мантисс, результат нормализуется и округляется. Эти операции выполняются так же, как и при сложении или вычитании. Необходимо учесть, что при нормализации может возникнуть переполнение или потеря значимости порядка.

**

Теперь рассмотрим алгоритм деления (рис. К.11). Как и ранее, первый этап — анализ операндов на равенство нулю. Если нулю равно делимое, то результату сразу присваивается значение 0. Если же нулю равен делитель, то в зависимости от конкретной реализации АЛУ результату может быть присвоено значение "бесконечность" с соответствующим знаком или сформирован сигнал арифметической ошибки.

Следующий этап — вычитание кода порядка делителя из кода порядка делимого. При этом получится несмещенный код разности, который нужно скорректировать — сложить с кодом смещения. После завершения операций с порядком результата проверяется, не возникло ли переполнение порядка или потеря значимости.

Следующий этап — деление мантисс. За ним следуют обычные операции нормализации и округления.

### *Точность выполнения операций*

#### *Дополнительные разряды*

Перед началом выполнения любых арифметических операций операнды загружаются в регистры АЛУ. Регистры, предназначенные для работы с мантиссами, могут иметь большую разрядность, чем поле мантиссы, предусмотренное форматом представления, плюс один неявно заданный старший разряд. Как правило, дополнительные разряды размещаются справа, т.е. имеют вес, меньший веса самого младшего разряда представления. При загрузке регистров эти разряды заполняются кодом 0.

Ниже представлены примеры, которые помогут вам понять смысл использования этих дополнительных разрядов. Будем считать, что для представления чисел используется формат, регламентированный стандартом IEEE — мантисса имеет длину 24 разряда, включая неявный старший разряд слева от разделительной точки. Два "соседних" числа X и Y, отличающиеся на очень малую величину, соответственно равны: X = 1.00...00х21, а Y = 1.11...11х20. При вычитании меньшего числа из большего мантисса меньшего числа должна быть сдвинута на один разряд вправо для того, чтобы выровнять порядки операндов. При этом число Y теряет младший разряд, и в результате получим разность, равную 2-22. При использовании дополнительных разрядов младший разряд мантиссы числа Y не теряется, и получается результат 2-23 который вдвое меньше предыдущего.

Примеры:

Основание характеристики равно 2, дополнительные разряды отсутствуют

|  |
| --- |
| *х = 1.000…..00x21* |
| *-у = 0.111…..11х21* |
| *z = 0.000…..01x21* |
| *= 1.000…..00x2-22* |

Основание характеристики равно 2 используются дополнительные разряды

|  |
| --- |
| *х = 1.000…..00 0000x21* |
| *-у = 0.111…..11 1000х21* |
| *z = 0.000…..01 1000x21* |
| *= 1.000…..00 0000x2-23* |

#### *Округление*

Результат любой операции над мантиссами операндов, как правило, формируется в регистре АЛУ, имеющем большую разрядность, чем предусмотрено форматом хранения. Поэтому при сохранении результата необходимо тем или иным способом выполнить его округление.

Стандартом IEEE предусматривается четыре альтернативных подхода к выполнению округления:

* округление до ближайшего числа, которое можно представить в используемом формате;
* округление до +∞;
* округление до -∞;
* округление до нуля.

#### *Особенности выполнения арифметических операций в соответствии со стандартом IEEE*

Стандартом IEEE 754 регламентируется не только формат с плавающей точкой для представления чисел, но и определенные правила выполнения арифметических операций над ними, что позволяет получать одинаковые результаты при реализации вычислительных алгоритмов на разных аппаратных и программных платформах. Один из аспектов стандарта, касающийся технологии округления, мы уже рассматривали выше. В этом разделе мы остановимся на трех других аспектах — операциях с бесконечно большими величинами, необычными числовыми величинами (типа NaN) и ненормализованными числами

Бесконечно большие величины

Арифметика бесконечных величин трактуется как частный случай арифметики вещественных чисел, в котором бесконечно большая величина интерпретируется следующим образом:

-∞ < (любая конечная числовая величина ) < ∞ .

За исключением особых случаев, которые будут рассмотрены в последующих разделах, любая арифметическая операция, в которой в качестве операнда выступает бесконечно большая величина, дает результат, соответствующий общепринятым правилам. Например:

|  |  |
| --- | --- |
| *5 + (+∞) = +∞* | *5 / (+∞) = +0* |
| *5 - (+∞) = -∞* | *(+∞) + (+∞) = +∞* |
| *5 + (-∞) = -∞* | *(-∞) + (-∞) = -∞* |
| *5 - (-∞) = +∞* | *(-∞) - (+∞) = -∞* |
| *5 x (+∞) = +∞* | *(+∞) - (-∞) = +∞* |

**Простые и сигнализирующие значения типа NaN**

Значения типа NaN представляют, по существу, не числа, а символы, "втиснутые" в формат с плавающей точкой. Различается два подтипа NaN-значений — сигнализирующее и простое. Появление в качестве операнда арифметической операции сигнализирующего NaN-значення генерирует исключительную ситуацию. Сигнализирующими NaN-значениями можно заполнять неинициализированные переменные и использовать их в специализированных арифметических расширениях, не вписывающихся в стандарт. В последнем случае перенос приложения на другую платформу, соответствующую стандарту, позволит сразу же обнаружить в нем нестандартные компоненты и внести необходимые изменения. Простые NaN-значения передаются далее по цепочке вычислительных процедур, не вызывая генерации исключительной ситуации. Операции, в которых в качестве результата может быть сформировано простое NaN-значение, перечислены в табл..6.

Оба подтипа NaN-значений представляются в формате с плавающей точкой одинаково — поле порядка заполнено единицами, а в поле мантиссы присутствует код, отличный от всех нулей, который стандарт не регламентирует. Этот код зависит от конкретной реализации вычислительной системы, и именно его можно использовать для того, чтобы различать обычные и сигнализирующие NaN-значения.

Таблица 6. Операции, порождающие простое NaN-значение

|  |  |
| --- | --- |
| *Операция* | *Условие формирования NaN-значения* |
| *Любая* | *Одним из операндов является сигнализирующее NaN-значение* |
| *Сложение или вычитание* | *Сложение или вычитание бесконечно больших/малых величин*  *( +∞ ) + ( -∞ )*  *( -∞ ) + ( +∞ )*  *( +∞ ) - ( +∞ )*  *( -∞ ) - ( -∞ )* |
| *Умножение* | *0х∞* |
| *Деление* | *0/0 или ∞/∞* |
| *Вычисление остатка* | *X MOD 0 или ∞MOD Y* |
| *Вычисление корня квадратного* | *, при х < 0* |

**Ненормализованные числа**

В формате с плавающей точкой, регламентированном стандартом IEEE 754, предусматривается возможность использования ненормализованной мантиссы для того, чтобы можно было обрабатывать ситуации потери значимости порядка. Когда порядок результата оказывается очень маленьким (большой по абсолютной величине порядок со знаком "минус"), этот результат можно денормализовать, сдвигая мантиссу вправо и одновременно увеличивая значение порядка до тех пор, пока число не будет "втиснуто" в разрядную сетку формата хранения

**Порядок выполнения работы**

Изучить теоретический материал и выполнить задания 2-3 в соответствии со своим вариантом

### *Задание 2*

Написать по шагам последовательность действий, реализующих *Операции сложения, вычитания, умножения и деления с плавающей* *точкой*  над двумя числами.

### *Задание 3*

Написать по шагам последовательность действий, реализующих *методы укоренного выполнения операций*  над двумя числами.

### *Варианты заданий*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *№ ВАРИАНТА* | *ЦЕЛЫЕ ЧИСЛА* | | *ЧИСЛА С ПАВАЮЩЕЙ ТОЧКОЙ* | |
| *Число А* | *Число В* | *Число А* | *Число В* |
| **1** | +56 | +167 | -12.000 | +0.50 |
| **2** | +172 | -45 | -0.875 | +54.375 |
| **3** | -82 | -146 | -15.750 | +0.250 |
| **4** | -224 | -75 | -0.625 | -52.000 |
| **5** | -94 | -230 | +22.50 | -0.875 |
| **6** | +142 | +22 | +0.375 | -45.750 |
| **7** | +85 | +234 | +59.250 | -0.625 |
| **8** | +156 | +47 | +0.125 | +34.250 |
| **9** | -67 | -223 | +45.250 | +0.375 |
| **10** | -175 | -54 | +0.375 | +43.50 |
| **11** | +45 | -156 | -39.50 | +0.875 |
| **12** | +228 | +76 | -0.625 | +34.750 |
| **13** | +202 | +79 | -38.750 | -0.375 |
| **14** | +39 | +159 | -0.875 | -32.000 |
| **15** | -176 | +34 | +49.000 | -0.375 |
| **16** | -45 | -229 | +0.625 | -58.50 |
| **17** | +185 | -81 | +46.50 | -0.750 |
| **18** | -48 | -52 | -0.375 | -44.875 |
| **19** | -180 | -27 | -56.250 | +0.50 |
| **20** | -97 | +183 | -13.000 | +0.875 |
| **21** | +236 | +58 | -0.875 | +18.50 |
| **22** | +58 | -239 | +76.750 | +0.625 |
| **23** | +248 | -29 | +0.375 | -45.50 |
| **24** | -67 | +139 | +61.875 | -0.50 |
| **25** | -184 | +78 | +0.50 | -47.125 |
| **26** | -103 | +74 | -29.625 | -34.750 |
| **27** | -99 | -166 | -38.750 | +0.250 |
| **28** | +93 | -175 | +0.250 | +70.625 |
| **29** | +208 | -74 | +57.375 | -0.50 |
| **30** | +49 | +183 | +0.125 | -51.875 |