# ИСКЛЮЧЕНИЯ ПРИ ДЕЛЕНИИ Приложение В

## В.1 ОСНОВНЫЕ ПРАВИЛА ДЕЛЕНИЯ

В наборе команд для процессоров семейства ADSP-2100 имеются две команды, позволяющие реализовать невосстанавливающий алгоритм деления. В качестве операндов этих команд берутся беззнаковые числа или числа в формате дополнительного кода, а результат представляет собой их усеченное до 16 бит частное, получаемое после соответсвующих вычислений в течение 16 циклов. Данные команды (примитивы деления) дают правильные результаты для большинства чисел и видов операций.Однако, могут возникать определенные ситуации, когда в полученном результате не будет доставать одного самого младшего бита. О таких ситуациях и будет рассказано в данном приложении, в котором будут предложены альтернативные решения для получения правильного результата.

16-разрядное частное с фиксированной точкой для двух беззнаковых чисел вычисляется путем выполнения команды DIVQ 16 раз. При делении знаковых чисел сначала используется команда DIVS, а затем 15 раз выполняется команда DIVQ. Независимо от типа выполняемого деления оба входных операнда должны быть в одном и том же формате (знаковом или беззнаковом) и давать результат того же формата.

Эти две команды используются для выполнения по условию невосстанавливающего алгоритма деления сложением/вычитанием. Как можно видеть из названия этого алгоритма, он работает за счет добавления или вычитания делителя к/из делимого. Выполняемая на каждом шаге операция (сложения или вычитания) зависит от полученного на предыдущем шаге бита частного. В результате каждой операции сложения/вычитания получается новый частный остаток, который используется на следующем шаге.

Слово "невосстанавливающий" отражает тот факт, что последний остаток получается неправильным. При восстанавливающем алгоритме возможно на любом шаге взять частное этого шага, умножить его на делитель и добавить частный остаток этого же шага, чтобы восстановить делимое. При невосстанавливающем алгоритме необходимо дважды добавить делитель к частному остатку в случае, когда бит частного,полученный на предыдущем шаге, является нулем. Для вычисления остатка проще использовать умножитель, чем АЛУ.

### В.1.1 Деление знаковых чисел.

Деление знаковых чисел начинается с записи 16-разрядного делителя в регистр X (AXO, AX1, AR, MR2, MR1,MR0,SR1 или SR0). 32-разрядное делимое должно храниться в двух отдельных 16-разрядных регистрах, при этом 16

младших бит должны храниться в регистре AY0, в то время как 16 старших - либо в AY1 либо в AF.

Примитив DIVS выполняется только один раз с правильно подобранными операндами (DVIVS AY1, AXO) и вычисляет знак частного. Для определения знакового бита частного, над знаковыми битами каждого операнда производится операция исключающего ИЛИ. Все 32 бита делимого сдвигаются влево на 1 бит. 15 младших бит делимого, к которым добавляется только что определенный знаковый бит, записываются в регистр AY0, а младшие 15 бит старшего слова, к которым добавляется самый старший бит младшего слова, записываются в регистр AF.

Затем для получения частного выполняется 15 команд DIVQ. Эта операция будет описана ниже.

### B.1.2 Деление беззнаковых чисел

Деление беззнаковых чисел выполняется так же, как и знаковых,за исключением того, что первой выполняемой командой тоже является команда DIVQ, а не DIVS. Старшее слово (16 бит) делимого должно храниться в регистре AF, в то время как бит AQ в регистре ASTAT должен до начала деления быть установлен равным нулю.

Бит AQ регистра ASTAT используется командой DIVQ, чтобы определить, должно ли делимое добавляться или вычитаться из частного остатка, хранящегося в регистре AF или AY0. Когда AQ равен нулю, производится вычитание. Новое значение бита AQ находится в результате выполнения операции исключающего ИЛИ над самым старшим битом делителя и самым старшим битом делимого. 32-разрядное делимое сдвигается на 1 бит влево, причем инвертированное значение бита AQ становится самым младшим битом.

### B.1.3 Форматы выходного результата

Как и при умножении,формат результата деления зависит от формата входных операндов. Используемая при делении логика была специально создана для максимально эффективной работы с дробными числами, которое наиболее часто использутся во всех операциях цифровой обработки сигналов, где требуются вычисления с фиксированной точкой. Один бит перед двоичной точкой знакового дробного числа используется как знак, в то время как 15 битов (или в вычислениях с двойной точностью, 31 бит) справа от него содержат информацию об абсолютном значении числа.

# ИСКЛЮЧЕНИЯ ПРИ ДЕЛЕНИИ Приложение B

Если делимое дано в формате M.N. (M бит до двоичной точки и N бит после нее), а делимое в формате O.P., частное будет иметь формат (M-O+1).(N-P-1). Как можно видеть, при делении числа в формате 1.31 на число в формате 1.15, полученное частное будет иметь формат (1-1+1).(31-15-1), т.е. 1.15.

Перед выполнением деления двух чисел следует убедиться, что получаемый формат частного будет правильным. Например, при попытке деления числа в формате 32.0 на число в формате 1.15, получится результат в формате (32-1+1).(0-15-1), т.е. 32.-16. Такое число не может быть представлено в 16-разрядном регистре !

Вам следует убедиться не только в правильности формата выходного результата, но и в том, не произошло ли переполнения. Даже когда при делении двух чисел получается число в разрешенном формате, остается вероятность переполнения этого числа, из-за чего оно не будет удовлетворять требованиям к результату деления. Например, при делении числа в формате 16.16 на число в формате 1.15 результат будет иметь разрешенный формат (16-1+1).(16-15-1), т.е. 16.0. Теперь предположим, что делимое равно 16384 (0x4000), а делитель равен .25 (0x2000), тогда их частное будет равно 65536, что не удовлетворяет формату 16.0. При данной операции произойдет переполнение, и будет выдан ошибочный результат.

Входные операнды могут проверяться на возможное переполнение до начала операции деления. Переполнение происходит, когда 16 старших бит делимого превышают по своему абсолютному значению делитель.

## B.1.4 Деление целых чисел

Особым случаем деления, который требует повышенного внимания, является деление целых чисел. Может возникнуть ситуация, когда желательно разделить одно целое число на другое, и при этом получить в результате также целое число. Ясно, что при делении целого числа на другое целое число, результат будет иметь неправильный формат : (32-16+1).(0-0-1), т.е. 17.-1.

Чтобы получить целое частное, делимое должно подвергнуться сдвигу на один бит влево, оказавшись, таким образом, в формате 31.1. Результат деления будет иметь формат (31-16+1).(1-0-1), т.е. 16.0. Следует убедиться, что в результате сдвига влево не были потеряны значимые биты, так как если это произошло, то будет получен неправильный результат.

# Приложение В ИСКЛЮЧЕНИЯ ПРИ ДЕЛЕНИИ

## В.2 УСЛОВИЯ ВОЗНИКНОВЕНИЯ ОШИБОК

Несмотря на то, что в большинстве случаев примитивы деления процессоров семейства ADSP-2100 работают правильно, имеется два случая, когда могут генерироваться неверные или недостаточно точные результаты. Первый случай представляет собой деление знакового числа на отрицательное. При попытке использовать отрицательное число в качестве делителя, в частном возможна ошибка в самом младшем бите. Второй случай относится к делению беззнаковых чисел, когда делитель превышает 0x7FFF. В случае делителя большего, чем 0x7FFF, при делении беззнакового числа на беззнаковое, будет сгенерирован неверный результат.

### В.2.1 Ошибка при отрицательном делителе

Результат деления на отрицательное число будет, как правило, иметь ошибку в самом младшем бите. Алгоритм деления, реализуемый процессорами семейства ADSP-2100 не совсем правильно компенсирует формат отрицательных чисел в дополнительном коде, что и вызывает данную неточность вычислений.

Здесь имеется одно исключение. Если в результате деления получается число 0x8000, то оно будет представлено правильно.

Эту ошибку можно компенсировать различными способами. Но до того как менять какой-либо код, следует определить, будет ли ошибка в самом младшем бите столь значима для решаемой задачи. В некоторых случаях эта ошибка слишком мала, чтобы быть значимой. Если же требуются только точные результаты, прибегните к одному из двух возможных решений.

Первое из них предлагает избежать деления на отрицательное число. Просто возьмите абсолютное значение отрицательного делителя, измените знак полученного частного на противоположный после деления, и вы получите правильный результат.

Другим способом может быть проверка результата умножением полученного частного на делитель. Сравните результат умножения с делимым и, если они не совпадают более, чем на значение делителя, увеличьте частное на единицу.

### B.2.2 Ошибка при делении беззнаковых чисел

Результат беззнакового деления может оказаться неправильным в случае, когда делимое превышает 0x7FFF. Не следует пытаться разделить одно беззнаковое число на другое, когда самый старший бит делителя равен единице. Если все же произвести такое деление необходимо, оба операнда должны быть сдвинуты вправо на 1 бит. Сдвиг позволит сохранить правильное расположение результата.

Сдвиг обоих операндов может дать ошибку в один самый младший бит в частном. Эта проблема может разрешаться за счет умножения частного на первоначальный (до сдвига вправо) делитель. Вычитание полученного при умножении значения из первоначального делимого даст ошибку. Если эта ошибка превышает делитель, увеличьте частное на единицу, если ошибка отрицательна, уменьшите частное на единицу.

## B.3 ПРОГРАМНОЕ РАЗРЕШЕНИЕ ТРУДНОСТЕЙ, ВОЗНИКАЮЩИХ ПРИ ДЕЛЕНИИ

Каждая из выше названных проблем может разрешаться с помощью программных средств. Далее будет приведен модуль программы *divide_ solution,* который может использоваться при делении двух знаковых или беззнаковых чисел для получения правильного частного или в условиях, когда возникают ошибки.

Кроме разрешения уже отмеченных выше проблем, в данном модуле реализована проверка переполнения при делении и вычисляется остаток после деления.

Так как не все задачи требуют полной проверки ошибок, программа создавалась с расчетом, чтобы пользователь мог избежать выполнения ненужных проверок. Последнее снижает требования к необходимому объему памяти и увеличивает скорость выполнения программы.

Модуль *signed_div* предусматривает, что 32- разрядное делимое записано в регистрах AY1 и AY0, а делитель - в регистре AX0. В результате выполнения подпрограммы либо частное содержится в регистре AR, а остаток в регистре MR0, либо устанавливается флаг переполнения. Выполнение всей подпрограммы занимает, самое большое, 27 циклов. В случае ситуации ошибки, возвращение в главную программу произойдет быстрее. Первые две команды записывают делимое в регистры MR, абсолютное значение старшего слова делимого - в AF, а абсолютное значение делителя - в AR.

Блок программы под меткой *test_1* проверяет деление на 0x8000. При попытке взять абсолютное значение числа 0x8000 происходит переполнение. Если

(в результате нахождения абсолютного значения делителя) устанавливается флаг AV, частное находится в AY1. Но если AY1 равно 0x8000, может произойти ошибка, поэтому после взятия из AY1 числа с противоположным знаком, флаг переполнения проверяется снова. Если этот флаг по-прежнему установлен, выполняется возвращение в вызывающую программу, если нет, вычисляется остаток. Если нет необходимости проверять, не равен ли делитель 0x8000, данный блок программы может пропускаться.

Блок программы под меткой *test_2* проверяет наличие условия переполнения при делении. Для этого из абсолютного значения старшего слова делимого вычитается абсолютное значение делителя. Если делитель меньше делимого, очень вероятно, что произойдет переполнение. Если делимое и делитель равны по абсолютному значению, но отличаются по знаку, результат будет равен 0x8000, и данный особый случай проверяется, как рассказывалось выше. Если для решения вашей задачи не требуется проверки переполнения, данный блок можно пропустить. Если вы решите пропустить блок *test_2,* не забудьте изменить адрес в команде JUMP в блоке *test_1* на *do_divs,* вместо *test_2.*

После проверки условий возникновения ошибок начинается непосредственное выполнение операции деления. Так как абсолютное значение делителя было записано в регистр AR, этот регистр используется как операнд X для команды DIVS. Затем следуют 15 команд DIVQ, которые вычисляют остальные биты частного. Правильный знак частного определяется на основание флага AS в регистре ASTAT. Так как первоначальное делимое содержится в регистре MR, остаток определяется путем операции умножения с вычитанием. Из содержимого регистра MR вычитается произведение частного и делетеля, полученный остаток записывается в регистр MR0.

Последним шагом перед возвращением в главную программу является очистка содержимого регистра ASTAT, который может содержать флаг переполнения, сгенерированный в ходе выполнения деления.

Подпрограмма *unsigned_div* очень напоминает программу *signed_div.* Старшее слово делимого загружается а регистры MR1 и AF, младшее слово делимого - в регистр MR0, а делитель содержится в AR. Так как деление беззнаковых чисел с большим делителем (>0x7FFF) запрещено, самый старший бит делимого проверяется на это условие. Если этот бит содержит единицу, устанавливается флаг переполнения и происходит возвращение в вызывающую программу. В остальных случаях блок *test_11* проверяет стандартные условия возникновения переполнения.

В блоке *test_11* производится вычитание делителя из старшего слова делимого. Если при этом получается результат меньше нуля, то операция деления продолжается, в обратном случае устанавливается флаг переполнения. Если

желательно не выполнять *test_11,* не забудьте изменить адрес в команде JUMP в блоке *test_10* на *do_divs.*

Непосредственное деление беззнакового числа на беззнаковое начинается с очистки бита AQ в регистре ASTAT, после чего 16 раз выполняется команда DIVQ. Остаток вычисляется после того, как регистр MR2 устанавливается равным 0. Это необходимо, так как значение в регистре MR1 автоматически дополняется по знаку в регистр MR2. Кроме того, должно быть выполнено умножение с беззнаковым переключением. Чтобы убедиться, что флаг переполнения сброшен, содержимое регистра ASTAT очищается перед возвращением в главную программу.

В обеих подпрограммах для вычисления остатка требуется только один дополнительный цикл, так что вам вряд ли понадобится удалять этот блок в целях увеличения скорости программы. Если изменение содержимого регистров умножителя вызывает какие-либо проблемы, удалите команду умножения с вычитанием непосредственно перед возвращением в главную программу, а также команды передачи данных в регистры MR0 и MR1 в первых двух многофункциональных командах. Убедитесь, что вы также удалили команду MR2=0 в подпрограмме *unsigned_div.*

```
.MODULE/ROM          Divide_solution ;
{
Данный модуль может использоваться для получения правильных результатов
при использовании примитивов деления в процессорах семейства ADSP-2100. Код
данной программы организован по блокам. Для обработки всех условий
возникновения ошибок может использоваться весь модуль целиком, в целях
увеличения скорости выполнения программы, отдельные блоки могут быть
удалены..

    Точки входа
    signed_div - вычисляет 16 - разрядное знаковое частное
    unsigned_div - вычисляет 16 - разрядное беззнаковое частное

    Параметры вызова

    AX0=16 -разрядный делитель
    AY0=младшие 16 бит делимого
    AY1=старшие 16 бит делимого

    Возвращаемые значения
    AR=16 - разрядное яастное
    MR0=16 - разрядный остаток

    Изменяемые регистры
    AX0, AX1, AR, AF, AY0, AY1, MR, MY0

    Время вычисления : 30 циклов (максимум)
}
```

# Приложение В ИСКЛЮЧЕНИЯ ПРИ ДЕЛЕНИИ

```
. ENTRY signed_div, unsigned_div ;
signed_div :    MR0=AY0, AF=AX0 + AY1;{ абсолютное значение }
                                        {делителя ,проверка не равны ли }
                MR1=AY1, AR=ABS AX0; { делитель и делимое по }
                                        { абсолютному значению }
test_1 :        IF NE JUMP test_2 ;     { если делитель не равен 0 }
                                        { переход к test_2 }
                ASTAT=0x4 ;             { деление на 0, переполнение }
                RTS ;                   { возвращение в вызывающую }
                                        { программу }
test_2 :        IF NOT AV JUMP test_3; { если делитель 0x8000, то част-}
                                        { ное равно -AY1 }
                IF NOT AV JUMP recover_sign ;
                ASTAT=0x4 ;             { 0x8000 деление на 0x8000,}
                RTS ;                   { отсюда переполнение }
test_3 :        AF=PASS AF;             { проверка на переполнение при делении}
                IF NE JUMP test_4;      { не равно, переход к test_4 }
                AY0=0x8000 ;            {частное равно -1 }
                ASTAT=0x0 ;             { очистка бита AS в ASTAT }
                JUMP recover_sign ;     { вычисление остатка }

test_4 :        AF=ABS MR1;     { нахождение абсолютного значения делимого }
                AR=ABS AX0;             { восстановление бита AS в ASTAT }
                AF=AF-AR ;              {проверка на переполнение }
                IF LT JUMP do_divs ;    { если делитель >делимого}
                ASTAT=0x4 ;             { переполнение при делении }
                RTS ;
do_divs :       DIVS AY1, AR ; DIVQ AR ; { вычисление знака частного }
                DIVQ AR ; DIVQ AR ;
                DIVQ AR ; DIVQ AR ;
                DIVQ AR ; DIVQ AR ;
                DIVQ AR ; DIVQ AR ;
                DIVQ AR ; DIVQ AR ;
                DIVQ AR ; DIVQ AR ;
                DIVQ AR ; DIVQ AR ;
recover_sign :  MY0=AX0, AR=PASS AY0 ;  {частное помещается в AR}
                IF NEG AR =-AY0 ;       {восстановление знака, если нужно }
                MR=MR - AR*MY0(SS) ;    {вычисление остатка, делимое }
                                        {отрицательно }
                RTS ;                   {возвращение в вызывающую }
                                        { программу}
```

# ИСКЛЮЧЕНИЯ ПРИ ДЕЛЕНИИ Приложение В

```
unsigned_div :  MRE=AY0, AF=PASS AY1; {помещение старшего слова }
                                      {делимого в регистр AF}
                MR1=AY1, AR=PASS AX0; {самый старший бит}
                                      {установлен ?}
test_10 :       IF GT JUMP test_11 ;  {нет, проверка переполнения }
                ASTAT=0x4 ;           {да, установка флага переполнения}
                RTS ;                 {возвращение в вызывающую}
                                      {программу }
test_11 :       AR=AY-AX0 ;           {делитель<делимого?}
                IF LT JUMP do_divq ;  {да,перейти к do_divq }
                ASTAT=0x4 ;           {установка флага переполнения}
                RTS ;

do_divq :       ASTAT=0 ;             {сброс флага AQ}
                DIVQ AX0 ; DIVQ AX0 ; {выполнение деления}
                DIVQ AX0 ; DIVQ AX0 ;
                DIVQ AX0 ; DIVQ AX0 ;
                DIVQ AX0 ; DIVQ AX0 ;
                DIVQ AX0 ; DIVQ AX0 ;
                DIVQ AX0 ; DIVQ AX0 ;
                DIVQ AX0 ; DIVQ AX0 ;
                DIVQ AX0 ; DIVQ AX0 ;

uremainder :    MR2=0 ;               {предыдущая установка MR0 и MR1 }
                MY0=AX0, AR=PASS AY0; {делитель в MY0, частное в AR }
                MR=MR - AR*MY0(UU) ; {определение остатка}
                RTS ;                 {возвращение в вызывающую}
                                      {программу}

.ENDMOD ;
```