Тема:

CRC (Cyclic Redundancy Codes – Циклические Избыточные Коды)

Выполнил:

Студент 915 группы
Морозов Александр

Москва , 2003 г.

1.Введение: обнаружение ошибок

Методы обнаружения ошибок предназначены для выявления повреждений сообщений при их передаче по зашумленным каналам (вносящих эти ошибки). Для этого передающее устройство создает некоторое число, называемое контрольной суммой и являющееся функцией сообщения, и добавляет его к этому сообщению.

Приемное устройство, используя тот же самый алгоритм, рассчитывает контрольную сумму принятого сообщения и сравнивает ее с переданным значением. Например, если мы для расчета контрольной суммы используем простое сложение байтов сообщения по модулю 256,то может возникнуть примерно следующая ситуация. (Все числа примера десятичные.)

Сообщение**:** **6 23 4**

Сообщение с контрольной суммой**: 6 23 4 33**

Сообщение после передачи**: 6 27 4 33**

Как Вы видите, второй байт сообщения при передаче оказался измененным с 23 на 27.Приемник может обнаружить ошибку, сравнивая переданную контрольную сумму (33) с рассчитанной им самим:6 +27 +4 =37.Если при правильной передаче сообщения окажется поврежденной сама контрольная сумма, то такое сообщение будет неверно интерпретировано, как искаженное. Однако, это не самая плохая ситуация. Более опасно, одновременное повреждение сообщения и контрольной суммы таким образом, что все сообщение можно посчитать достоверным. Недостаток данного алгоритма в том, что он слишком прост. Если произойдет несколько искажений, то в 1 случае из 256 мы не сможем их обнаружить. Например:

Сообщение**:** **6 23 4**

Сообщение с контрольной суммой**: 6 23 4 33**

Сообщение после передачи**: 8 20 5 33**

Для повышения надежности мы могли бы изменить размер регистра с 8 битного на 16 битный (то есть суммировать по модулю 65536 вместо модуля 256),что скорее всего снизит вероятность ошибки с 1/256 до 1/65536. Хотя по-настоящему проблема может быть решена лишь заменой простого суммирования более сложной функцией, чтобы каждый новый байт оказывал влияние на весь регистр контрольной суммы.

Таким образом, мы сформулируем 2 требования для формирования надежной контрольной суммы:

Ширина: Размер регистра для вычислений должен изначально обеспечивать низкую вероятность ошибки (например,32 байтный регистр обеспечивает вероятность ошибки $1/2^{32}$).

Случайность: Необходим такой алгоритм расчета, когда каждый новый байт может оказать влияние на любые биты регистра.

2.Основная идея, заложенная в алгоритме CRC

Основная идея алгоритма CRC состоит в представлении сообщения виде огромного двоичного числа, делении его на другое фиксированное двоичное число и использовании остатка этого деления в качестве контрольной суммы. Получив сообщение, приемник может выполнить аналогичное действие и сравнить полученный остаток с "контрольной суммой" (переданным остатком). Приведем пример:

Предположим, что сообщение состоит из 2 байт (6,23), как в предыдущем примере. Их можно рассматривать, как двоичное число 0000 0110 0001 0111. Предположим, что ширина регистра контрольной суммы составляет 1 байт, в качестве делителя используется 1001, тогда сама контрольная сумма будет равна остатку от деления 0000 0110 0001 0111 на 1001.Хотя в данной ситуации деление может быть выполнено с использованием стандартных 32 битных регистров, в общем случае это неверно. Поэтому воспользуемся делением "в столбик" в двоичной системе счисления:

1001=9d делитель

0000011000010111 =0617h =1559d делимое
0000011000010111/1001=2d остаток
Хотя влияние каждого бита исходного сообщения на частное не столь существенно, однако 4 битный остаток во время вычислений может радикально измениться, и чем больше байтов имеется в исходном сообщении (в делимом), тем сильнее меняется каждый раз величина остатка. Вот почему деление оказывается применимым там, где обычное сложение работать отказывается.
В нашем случае передача сообщения вместе с 4 битной контрольной суммой выглядела бы (в шестнадцатеричном виде) следующим образом:06172,где 0617 – это само сообщение, а 2 – контрольная сумма. Приемник, получив сообщение, мог бы выполнить аналогичное деление и проверить, равен ли остаток переданному значению (2).

3. Полиномиальная арифметика

Все CRC алгоритмы основаны на полиномиальных вычислениях, и для любого алгоритма CRC можно указать, какой полином он использует. Что это значит?
Вместо представления делителя, делимого (сообщения), частного и остатка в виде положительных целых чисел, можно представить их в виде полиномов с двоичными коэффициентами или в виде строки бит, каждый из которых является коэффициентом полинома. Например, десятичное число 23 в шестнадцатеричной системе счисления имеет вид 17,в двоичном – 10111, что совпадает с полиномом:
1*x ^4 +0*x ^3 +1*x ^2 +1*x ^1 +1*x ^0
или, упрощенно:
x ^4 +x ^2 +x ^1 +x ^0
И сообщение, и делитель могут быть представлены в виде полиномов, с которыми, как и раньше можно выполнять любые арифметические действия; только теперь надо не забывать о “иксах”. Предположим, что мы хотим перемножить, например,1101 и 1011.Это можно выполнить, как умножение полиномов:
(x ^3 +x ^2 +x ^0)(x ^3 +x ^1 +x ^0)
=(x ^6 +x ^4 +x ^3+x ^5 +x ^3 +x ^2+x ^3 +x ^1 +x ^0)=
=x ^6 +x ^5 +x ^4 +3*x ^3 +x ^2 +x ^1 +x ^0
Теперь для получения правильного ответа нам необходимо указать, что **x** равен 2,и выполнить перенос бита от члена **3*x ^3** .В результате получим:
x ^7 +x ^3 +x ^2 +x ^1 +x ^0
А то, что **если** мы считаем, "X" нам не **известным**, то мы не можем выполнить перенос. Нам не известно, что **3*x ^ 3** – это то же самое, что и **x ^4 +x ^3**, так как мы не знаем, что **x** =2. В полиномиальной арифметике связи между коэффициентами не установлены и, поэтому, коэффициенты при каждом члене полинома становятся строго типизированными — коэффициент при **x ^ 2** имеет иной тип, чем при **x ^3** .
Если коэффициенты каждого члена полинома совершенно изолированы друг от друга, то можно работать с любыми видами полиномиальной арифметики, просто меняя правила, по которым коэффициенты работают. Одна из таких схем для нам чрезвычайно интересна, а именно, когда коэффициенты складываются по модулю 2 без переноса – то есть коэффициенты могут иметь значения лишь 0 или 1,перенос не учитывается. Это называется "полиномиальная арифметика по модулю 2".
Возвращаясь к предыдущему примеру, получим результат:
=x ^6 +x ^5 +x ^4 +x ^3 +x ^2 +x ^1 +x ^0

4. Двоичная арифметика без учета переносов

Сфокусируем наше внимание на арифметике, применяемой во время вычисления CRC. Фактически как операция сложения, так и операция вычитания в CRC арифметике идентичны операции "Исключающее ИЛИ"(eXclusive OR –XOR),что позволяет заменить 2 операции первого уровня (сложение и вычитание) одним действием, которое,

одновременно, оказывается инверсным самому себе. Определив действие сложения, перейдем к умножению и делению. Умножение, как и в обычной арифметике, считается суммой значений первого сомножителя, сдвинутых в соответствии со значением второго сомножителя. При суммировании используется CRC-сложение. Деление несколько сложнее, хотя вполне интуитивно понятно как его выполнять. Действия аналогичны простому делению в столбик с применением CRC-сложения.
Если число A получено умножением числа B, то в CRC арифметике это означает, что существует возможность сконструировать число A из нуля, применяя операцию XOR к число B, сдвинутому на различное количество позиций. Например, если A равно 0111010110,а B – 11,то мы может сконструировать A из B следующим способом:
0111010110
=.....11.
+....11....
+...11.....
+.11.......
Однако, если бы A было бы равно 0111010111,то нам бы не удалось составить его с помощью различных сдвигов числа 11. Поэтому что, как говорят в CRC арифметике, оно не делится на B.

5. Полностью рабочий пример

Чтобы выполнить вычисление CRC, нам необходимо выбрать делитель. Говоря математическим языком, делитель называется генераторным полиномом(generator polinomial), или просто полиномом, и это ключевое слово любого CRC алгоритма. Степень полинома W (Width – ширина, позиция самого старшего единичного бита) чрезвычайно важна, так как от нее зависят все остальные расчеты. Обычно выбирается степень 16 ил 32,т.к. как это облегчает реализацию алгоритма на современных компьютерах. Степень полинома – это действительная позиция старшего бита, например, степень полинома 10011 равна 4 а, не 5. Выбрав полином приступим к расчетам. Это будет простое деление (в терминах CRC арифметики) сообщения на наш полином. Единственное, что надо будет сделать до начала работы, так это дополнить сообщение W нулевыми битами. Итак, начнем:
Исходное сообщение**:** **1101011011**
Полином**:** **10011**
Сообщение, дополненное W битами**:11010110110000**
Теперь просто поделим сообщение на полином, используя правила CRC-арифметики.
11010110110000 / 10011 = 1100001010 (частное, оно никого не интересует**) +**
+1110 = остаток = контрольная сумма!
Как правило, контрольная сумма добавляется к сообщению и вместе с ним передается по каналам связи. В нашем случае будет передано следующее сообщение:**11010110111110.**
На другом конце канала приемник может сделать одно из равноценных действий:
1)Выделить текст собственно сообщения, вычислить для него контрольную сумму (не забыв при этом дополнить сообщение W битами), и сравнить ее с переданной.
2)Вычислить контрольную сумму для всего переданного сообщения (без добавления нулей), и посмотреть, получится ли в результате нулевой остаток.
Оба эти варианта совершенно равноправны. Однако отныне мы будем работать со вторым вариантом, которое является математически более правильным.
Таким образом, при вычислении CRC необходимо выполнить следующие действия:
1)Выбрать степень полинома W и полином G (степени W).
2)Добавить к сообщению W нулевых битов. Назовем полученную строку M'.
3)Поделим M' на G с использованием правил CRC арифметики. Полученный остаток и будет контрольной суммой.

6.Выбор полинома

Во первых, надо отметить, что переданное сообщение T является произведением полинома. Чтобы понять это, обратите внимание, что 1)последние W бит сообщения – это остаток от деления дополненного нулями исходного сообщения на выбранный полином, и 2)сложение равносильно вычитанию, поэтому прибавление остатка дополняет значение сообщения до следующего полного произведения. Теперь смотрите, если сообщение при передаче было повреждено, то мы получим сообщение T + E, где E – это вектор ошибки, '+ '– это CRC сложение (или операция XOR). Получив сообщение, приемник делит T + E на G. Так как T mod G =0,(T+E) mod G =E mod G. Следовательно, качество полинома, который мы выбираем для перехвата некоторых определенных видов ошибок, будет определяться набором произведений G, так как в случае, когда E также является произведением G, такая ошибка выявлена не будет. Следовательно, наша задача состоит в том, чтобы найти такие классы G, произведения которых будут как можно меньше похожи на шумы в канале передачи (которые и вызывают повреждение сообщения).Давайте рассмотрим, какие типы шумов в канале передачи мы можем ожидать.

Однобитовые ошибки. Ошибки такого рода означают, что E=1000...0000.Мы можем гарантировать, что ошибки этого класса всегда будет распознаны при условии, что в G по крайней мере 2 бита установлены в "1".Любое произведение G может быть сконструировано операциями сдвига и сложения, и, в тоже время, невозможно получить значение с 1 единичным битом сдвигая и складывая величину, имеющую более 1 единичного бит, так как в результате всегда будет присутствовать по крайней мере 2 бита.

Двухбитовые ошибки. Для обнаружения любых ошибок вида 100...000100...000 (то есть когда E содержит по крайней мере 2 единичных бита) необходимо выбрать такое G, которые бы не имело множителей 11,101,1001,10001,и так далее. В качестве примера приведем полином с единичными битами в позициях 15,14 и 1, который не может быть делителем ни одно числа меньше 1...1, где "..."32767 нулей.

Ошибки с нечетным количеством бит .Мы может перехватить любые повреждения, когда E имеет нечетное число бит, выбрав полином G таким, чтобы он имел четное количество бит. Чтобы понять это, обратите внимание на то, что 1)CRC умножение является простой операцией XOR постоянного регистрового значения с различными смещениями; 2)XOR – это всего-навсего операция переключения битов;
и 3)если Вы применяете в регистре операцию XOR к величине с четным числом битов, четность количеств единичные битов в регистре останется неизменной. Например, начнем с E=111 и попытаемся сбросить все 3 бита в "0" последовательным выполнением операции XOR с величиной 11 и одним из 2 вариантов сдвигов (то есть, "E=E XOR 011"и "E=E XOR 110").Это аналогично задаче о перевертывании стаканов, когда за одно действие можно перевернуть одновременно любые два стакана. Большинство популярных CRC полиномов содержат четное количество единичных битов.

Пакетные ошибки .Пакетная ошибка выглядит как E=000...000111...11110000...00, то есть E состоит из нулей за исключением группы единиц где-то в середине. Эту величину можно преобразовать в E=(00000...00)(1111111...111),где имеется z нулей в левой части и n единиц в правой. Для выявления этих ошибок нам необходимо установить младший бит G в 1.При этом необходимо, чтобы левая часть не была множителем G. При этом всегда, пока G шире правой части, ошибка всегда будет распознана.

Несколько популярных полиномов:

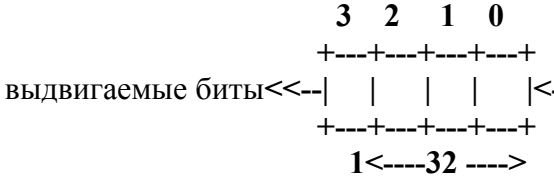
16 битные:(16,12,5,0)[стандарт "X25"]
(16,15,2,0)["CRC 16"]
32 битные:(32,26,23,22,16,12,11,10,8,7,5,4,2,1,0) [Ethernet]

8.Прямая реализация табличного алгоритма CRC

Для увеличения скорости обработки битов нам необходимо найти способ заставить алгоритм работать с блоками, большими, чем бит. Такими блоками могут быть

полубайты (4 бит), байты (8 бит), слова (16 бит) и длинные слова (32 бита), и даже более длинные фрагменты, если мы только сможем с ними справиться. С полубайтами лучше не связываться, так как они не выровнены на границу байт .Для увеличения скорости работы программы нам необходимо, чтобы информационные блоки были выровнены на границу байта, и, фактически, большинство табличных алгоритмов оперируют за один цикл именно с байтом. Для простоты обсуждения, давайте перейдем от 4 битного полинома к 32 битному.

Наш регистр будет выглядеть следующим образом, ячейки представляют собой байты, а полином будет состоять из 33 битов
(один подразумеваемый единичный бит и 32 "активных" бит)(W=32).



Старшие 8 бит 32 битового регистра (3 байт) имеют значение:
t7 t6 t5 t4 t3 t2 t1 t0 . Предположим также, что старшие 8 бит полинома
имеют значения **g7 g6 …g0** ,тогда к следующему циклу старший байт будет иметь следующее содержимое:

t6 t5 t4 t3 t2 t1 t0 ??
+
t7 *(g7 g6 g5 g4 g3 g2 g1 g0)

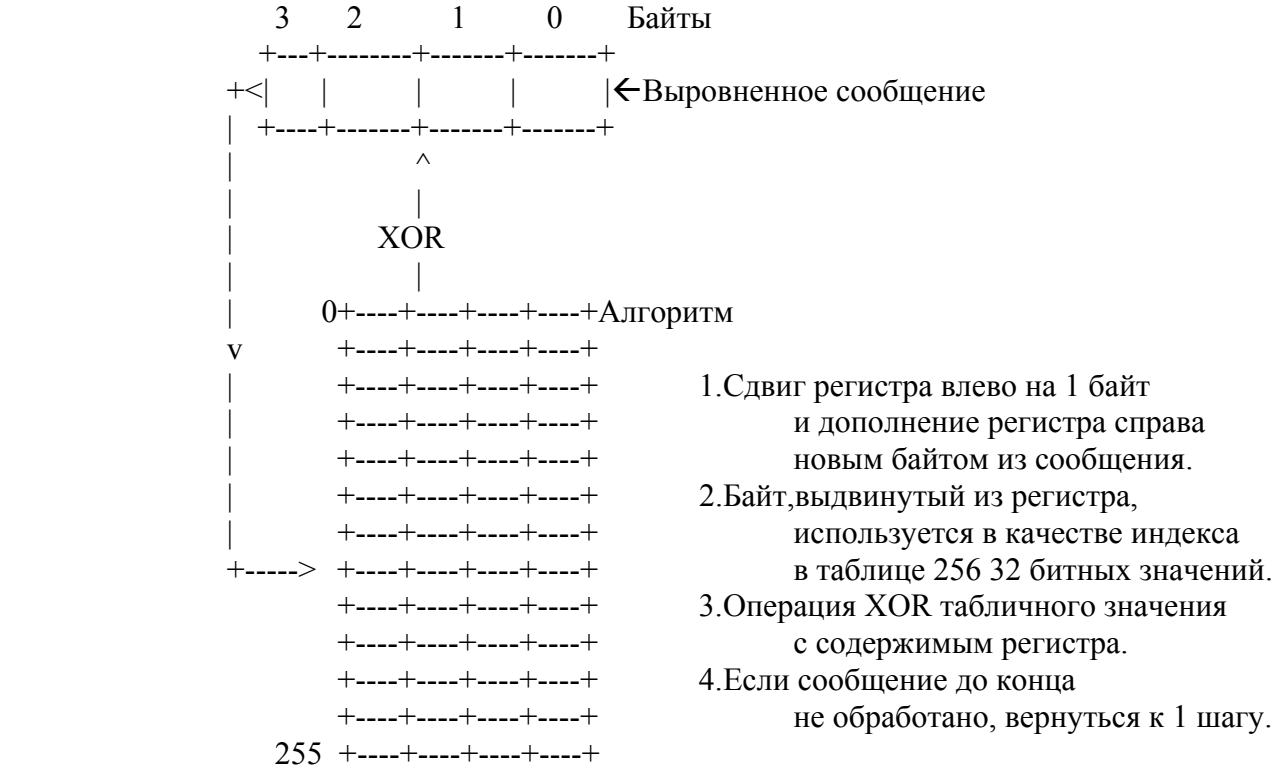
"Новый" старший бит (который будет определять, что произойдет в следующем цикле) теперь имеет значение **t6** + **t7*g7** .Обратите внимание, что с информационной точки зрения все, что требуется для вычисления значения "нового" старшего бита, содержится в 2 старших битах исходного старшего байт. Точно также, "следующий" старший бит может быть заранее вычислен, исходя их 3 старших битов **t7,t6** и **t5** .В общем случае величину старшего бита регистра на k цикле можно вычислить из k старших битов регистра. Получается следующая программа:

```
While (сообщение полностью не обработано)
  Begin
      Проверим старший байт регистра
      Рассчитаем на его основе контрольный байт
      Основываясь на значении контрольного байта,
      выполним операцию XOR нашего полинома с различными смещениями
      Сдвинем регистр на один байт влево,добавив справа новый байт из сообщения
      Выполним операцию XOR содержимого регистра с суммированным полиномом
  End
```

Оказывается, однако, что большую часть вычислений можно провести заранее, результаты собрать в таблицу. Тогда получим:

```
While (сообщение полностью не обработано)
 Begin
      Top =Старший_байт(Register);
      Register =(Register <<24)|Новый_байт_сообщения;
      Register =Register XOR Рассчитанная_таблица [Top ];
 End
```

Графически он выглядит так:



9.Зеркальный табличный алгоритм

Определение. Значение (или регистр) является отраженным (или обращенным),если все его биты поменялись местами относительно центра. Например:0101 является 4 битным отражением величины 1010.

Получилось так, что UART (эти аккуратные маленькие микросхемы, которые выполняют операции последовательного ввод /вывода) имеют привычку передавать каждый байт, начиная с наименее значащего бита (бита 0), и заканчивая наиболее значащим (битом 7), т.е. в обратной — в зеркальной последовательности. В результате специалистам по аппаратному обеспечению, занимающимся аппаратным расчетом CRC на уровне битов, пришлось выбрать такие алгоритмы, которые работают с зеркально отраженными байтами. Сами байты обрабатываются в своем естественном порядке, однако их биты отражены относительно центра байта: бит 0 теперь становится битом 7,бит 1 –битом 6 и так далее. Не следует считать, что работа таких алгоритмов возможна только на аппаратном уровне. На каком-то этапе они, по видимому, вышли на уровень программ, и кому-то пришлось написать код, который мог бы взаимодействовать с аппаратурой должным образом.

Итак:

1.Используется та же самая таблица, правда, содержимое каждой ее позиции зеркально обращено относительно центра позиции.

2.Используется то же самое исходное значение регистра, только так же отраженное.

3.Байты сообщения обрабатываются в том же порядке, что и раньше, то есть само сообщение не является зеркальным.

4.Сами байты сообщения не нужно зеркально отображать, так как все остальное уже было отражено!

По окончании вычислений регистр будет содержать зеркальное отражение конечного значения CRC (остаток).Независимо от того, имеет ли это смысл или не имеет, существование "зеркального" алгоритма в мире FTP сайтов приводит к тому, что половина реализаций расчета CRC использует его, другая половин нет.

10. Начальные и конечные значения

В довершение ко всему нами виденному, CRC алгоритмы различаются еще по 2 моментам:

1.по начальному значению регистра;

2.по значению, которое комбинируется по XOR с окончательным содержимым регистра.

Например, алгоритм "CRC 32"инициализирует регистр значением **0xFFFFFFFF** и выполняют операцию XOR окончательного значения также с величиной **0xFFFFFFFF** . Большинство CRC алгоритмов инициализируют регистр нулевым значением, однако некоторые предпочитают ненулевое. Теоретически (когда не делается никаких предположений относительно содержания сообщения) начальной значение не влияет на стойкость CRC, оно лишь является точкой отсчета, с которой алгоритм начинает работать. Однако, на практике некоторые сообщения более вероятны, чем другие, поэтому разумнее будет выбрать такое начальное значение регистра,которое бы не имело "слепых пятен". Под "слепым пятном" мы подразумеваем такую последовательность байтов сообщения, которые не приводят к изменению содержимого регистра.

11.Полное определение алгоритма

Итак, CRC алгоритмы различаются по:

1.степени полинома;

2.значению полинома;

3.начальному содержимому регистра;

4.обращаются ли биты каждого байта перед их обработкой;

5.требуется ли "проталкивать" байты сообщения через регистр, или они комбинируются по XOR с содержимым регистра, а затем делается поиск в таблице;

6.нужно ли обращать конечный результат (как в "зеркальной" версии);

7.значению, которое комбинируется по XOR с окончательным содержимым регистра.

Для того, чтобы в дальнейшем говорить об особенностях различных CRC алгоритмов, нам необходимо определить все эти пункты более точно. По этой причине в следующем разделе будет сделана попытка составить параметрическую модель CRC алгоритма.

12.Параметрическая модель алгоритма

Сейчас мы займемся составлением точной параметрической модели CRC алгоритма, которую, за неимением лучшего, мы будем называть моделью Rocksoft ™.

Данная модель будет рассмотрена исключительно с точки зрения функционирования, игнорируя при этом все детали реализаций. Основной целью будет создание способа точного описания различных частных алгоритмов CRC, вне зависимости от того, насколько сложно они реализованы. С этой точки зрения модель должна быть настолько простой и точной, насколько это возможно, и при этом как можно более понятной.

Модель CRC алгоритма Rocksoft ™ базируется на рассмотренном прямом табличном алгоритме. Однако, этот алгоритм должен быть параметризован так, чтобы отражать поведение различных реальных алгоритмов.

Для функционирования алгоритма в качестве "зеркального" добавим две логические переменные: одну, определяющую обращение поступающих байтов, другую, указывающую, нужно ли обращать окончательное значение контрольной суммы. Ограничивая "зеркальность" алгоритма только трансформацией на входе и выходе, мы избегаем ненужного усложнения при рассмотрении "зеркальных"и "не зеркального" алгоритмов.

Дополнительный параметр позволяет инициализировать регистр начальным

значением. Еще один параметр указывает на необходимость комбинации содержимого регистра с некоторым значением перед выдачей конечного результата.
Опишем теперь все параметры модели:

Name: Это имя, присвоенное данному алгоритму. Строковое значение.

Width: Степень алгоритма, выраженная в битах. Она всегда на единицу меньше длины полинома, но равна его степени.

Poly: Собственно полином. Это битовая величина, которая для удобств может быть представлена шестнадцатеричным числом. Старший бит при этом опускается. Например, если используется полином 10110,то он обозначается числом "06h".Важной особенностью данного параметра является то, что он всегда представляет собой не обращенный полином, младшая часть этого параметра во время вычислений всегда является наименее значащими бит ми делителя вне зависимости от того, какой –"зеркальный" или прямой алгоритм моделируется.

Init: Этот параметр определяет исходное содержимое регистр на момент запуска вычислений. Именно это значение должно быть занесено в регистр в прямой табличном алгоритме. В принципе, в табличных алгоритмах мы всегда может считать, что регистр инициализируется нулевым значением, а начальное значение комбинируется по XOR с содержимым регистра после N цикла. Данный параметр указывается шестнадцатеричным числом.

RefIn: Логический параметр. Если он имеет значение "False" ("Ложь"), байты сообщения обрабатываются, начиная с 7 бита, который считается наиболее значащим, а наименее значащим считается бит 0.Если параметр имеет значение "True"("Истина"), то каждый байт перед обработкой обращается.

RefOut: Логический параметр. Если он имеет значение "False"("Ложь"),то конечное содержимое регистр сразу передается на стадию XorOut,в противном случае, когда параметр имеет значение "True"("Истина"), содержимое регистра обращается перед передачей на следующую стадию вычислений.

XorOut: W битное значение, обозначаемое шестнадцатеричным числом. Оно комбинируется с конечным содержимым регистра (после стадии RefOut), прежде чем будет получено окончательное значение контрольной суммы.

Check: Это поле, собственно, не является частью определения алгоритма, и, в случае противоречия с ним предшествующих параметров, именно эти предыдущие параметры имеют наибольший приоритет. Данное поле служит контрольным значением, которое может быть использовано для слабой проверки правильности реализации алгоритма. Поле содержит контрольную сумму, рассчитанную для ASCII строки "123456789"(шестнадцатеричные значение "313233 …").

После определения всех этих параметров, наша модель может быть использована для точного описания особенностей каждого CRC алгоритма. Ниже приведен пример спецификации популярного варианта алгоритма "CRC 16" и "CRC 32":

Name :	"CRC 16"	“CRC 32”
Width :	16	32
Poly :	8005	04C11DB7
Init :	0000	FFFFFFFF
RefIn :	True	
RefOut :	True	True
XorOut :	0000	FFFFFFFF
Check :	BB3D	

13.Восстановление CRC

Алгоритм CRC построен так, чтобы вне зависимости от того, какой **бит** Вы меняете, результат вычисления **всегда** (хорошо, почти всегда)преображается совершенно неузнаваемо. Можно "корректировать" CRC лишь **после** изменения необходимых мне байтов.

Имеется последовательность байтов:

01234567*8901234567890123456* **78901234567890123456789012**

Вам необходимо изменить байт, выделенные подчеркиванием (с 9 по 26).

Кроме того, Вам понадобятся еще 4 байта (до 30й позиции) для построения такой последовательности, которая восстановит исходное значение CRC.

Когда Вы начинаете вычислять CRC 32 все идет прекрасно до байта в 9 позиции; в измененной же последовательности, начиная с этой точки, CRC меняется кардинально. Даже после 26 позиции, байты после которой изменений не претерпели, получить исходного значения CRC оказывается невозможно.

Вскоре Вы поймете, что следует предпринять в такой ситуации. В двух словах это можно описать следующим образом:

1.Необходимо вычислить значение CRC вплоть до 9 позиции и запомнить его.

2.Продолжить расчет CRC до 30 позиции включительно (байт ,которые Вы собираетесь менять, плюс 4 байта дополнительно)и также запомнить значение.

3.Рассчитать значение CRC для "новых" байтов, включая 4 дополнительных (то есть всего для 27 9+4=22 байтов), используя в расчетах величину, полученную в шаге 1, снова его запомнить.

4.Теперь мы имеем "новое" значение CRC. Однако, нам необходимо, чтобы CRC имело "старое" значение. Для расчета 4 дополнительных байтов необходимо "обратить" алгоритм.

14.Восстановление CRC-16

Для начала мы займемся восстановлением 16 разрядного CRC. Итак, мы сделали в последовательности все необходимые изменения, и теперь нам необходимо вернуть прежнее значение CRC. Мы знаем старое значение CRC (рассчитанное до изменения последовательности), а также его новое значение. Нам необходимо составить такую 2 байтную последовательность, которая позволит изменить текущее значение CRC в его прежнее состояние. Сначала мы рассчитаем CRC с учетом 2 неизвестных байт (назовем их "X" и "Y"). Предположим, что регистр имеет значения a1 и a0,а вдвигаемый байт равен 00. Возьмем последовательность "X Y". Байт обрабатываются слева направо. Предположим, регистр равен "a1 a0".Будем обозначать операцию XOR символом "+ ".

Обработаем 1 байт (“X”):

a0 + **X**	“старший” байт (1)
b1 b0	табличная последовательность для этого старшего байта
00 a1	регистр, сдвинутый вправо
00 + **b1 a1** + **b0**	результат операции XOR двух предыдущих строк

Теперь регистр содержит:**(b1)(a1** + **b0)**.

Обработаем 2 байт ("Y"):

(a1 + **b0)**+ **Y**	"старший" байт (2)
1 0	табличная последовательность для этого старшего байта
00 b1	регистр, сдвинутый вправо
00 +**1 b1** +**c0**	результат операции XOR двух предыдущих строк

В результате регистр будет иметь значение **(c1)(b1** + **c0)**.

Запишем эти вычисления в несколько ином виде:

a0 + **X =(1)**	указатель табличного значения **b1 b0**

a1 + **b0** + **Y** =**(2)** указатель табличного значения **c1 c0**
b1 + **c0** =**d0** новый младший байт регистра
c1 =**d1** новый старший байт регистра
(1) (2)

Теперь давайте переварим полученные результат. Мы хотим получить в регистре значение **d1 d0** (исходное CRC), зная содержимое регистра обработки измененной последовательности символов (значение **a1 a0**). Вопрос – какие 2 байта, или, другими словами, какие значения "X" и "Y" нам необходимо использовать при вычислении CRC? Начнем рассуждать с конца. **d0** должно быть равно **b1** + **c0** ,а **d1** =**c1** .Но нужно ли мне напоминать Вам о Таблице? А значение **c0** можно получить из слова **c0 c1** , ведь мы знаем значение **c1** .Следовательно нам потребуется процедура поиска А, если В нашли это слово, то уж постарайтесь запомнить и его индекс, с помощью которого мы сможем найти старшие группы байт, то есть величины **(1)**и **(2)**.
Но как же нам получить **b1 b0** ,несмотря на то, что мы нашли **c1 c0** ?Если **b1** + **c0=d0** ,то **b1=d0 _ c0** !Ну, а затем, найдя **b1** ,применим процедуру поиска для получения слова **b1 b0** .Теперь у нас есть все для вычисления значений "X" и "Y"!

15.Численный пример вычисления

Давайте теперь рассмотрим реальный числовой пример:
1. исходное значение регистра:**(a1=)DEh (a0=)ADh**
2. нужно получить: **(d1=)12h (d0=)34h**

Найдем в таблице для CRC 16 строку, начинающуюся с 12h. Это позиция 38h,которая содержит значение 12C0h.Это единственное такое значение, ведь, вспомните, мы рассчитывали каждую позицию в таблице для всех возможных величин старшей группы, всего их было 256.Теперь мы знаем, что **(2)=38** ,**c1=12** ,а **c0=C0** , следовательно **b1=C0+ +34=F4** ,и следующим шагом необходимо найти позицию, значение в которой начинается с F4h.Это оказалась позиция 4Fh содержащая F441h.Получаем,**(1)=4F** ,**b1=F4** , **b0=41** .Теперь у нас есть все, чтобы рассчитать "X" и "Y":

Y=a1 + b0 + (2)=DE + 41 + 38 =A7
X=a0 + (1)=AD+4F =E2

Следовательно, чтобы изменить значение регистра CRC 16 со значения DEAD на значение 1234 необходимы байты E2 A7 (именно в таком порядке).
Как Вы видите, чтобы восстановить CRC, Вы должны "просчитать" его в обратном порядке и запомнить все полученные значения. При реализации программы просмотра таблицы обратите внимание, что для процессоров фирмы Intel характерно запоминание байтов в обратном порядке (сначала младший, а затем старший байт).

Ссылки

[Griffiths87]Griffiths,G.,Carlyle Stones,G.,"The Tea Leaf Reader Algorithm:An Efficient Implementation of CRC 16 and CRC 32",Communications of the ACM,30(7), pp.617 620.
В данной статье описывается высокоскоростная табличная реализация CRC алгоритмов. Использованный способ слегка странен и может быть заменен алгоритмом Sarwate.
[Knuth81] Knuth,D.E.,"The Art of Computer Programming",Volume 2: Seminumerical Algorithms,Section 4.6.(Кнут Д.Е."Искусство программирования для ЭВМ",т.2 "Получисленные алгоритмы"– М..,"Мир",1977).