УДК 004.42

Мыцко Евгений Алексеевич
ФГАОУ ВО «Национальный исследовательский Томский политехнический университет»
Россия, Томск[1]
Магистрант
E-Mail: evgenrus70@mail.ru

Примеры программных реализаций алгоритмов вычисления контрольной суммы Cyclic redundancy check 32, совместимой с WinRAR, PKZIP, Ethernet

Аннотация. В данной работе рассмотрены конкретные примеры программных реализаций алгоритмов вычисления контрольной суммы cyclic redundancy check 32. На основе исходных кодов, имеющихся в свободном доступе, осуществлены программные реализации однобайтового, двухбайтового и четырёхбайтового матричного алгоритмов вычисления cyclic redundancy check 32. Для рассмотренных программных реализаций восстановлены описания алгоритмов вычисления контрольной суммы. Рассмотрены различия в программной реализации табличного и матричного алгоритмов вычисления cyclic redundancy check 32. Показано, что для программной реализации матричного алгоритма необходимо заменить основной цикл исходного кода из свободного доступа, а также изменить матрицу предвычисленных значений. Показана практическая значимость приведённых примеров программной реализации алгоритмов вычисления контрольной суммы cyclic redundancy check 32. На примере исходных кодов показано, что матричный алгоритм достаточно просто модифицировать, увеличив количество обрабатываемых байт за итерацию, а также для его реализации требуется значительно меньше памяти. Так, для реализации табличного алгоритма требуется 1 Кб памяти на хранение таблицы, в то время, как для матричного алгоритма всего 32 байта, для матричного двухбайтового – 64 байта, для четырёхбайтового – 128 байт.

Ключевые слова: контрольная сумма; программная реализация; табличный алгоритм; матричный алгоритм; cyclic redundancy check 32; исходный код; однобайтовый; двухбайтовый; четырёхбайтовый; данные.

Идентификационный номер статьи в журнале 140TVN314

[1] 634050, г. Томск, пр-кт Ленина, 30, ТПУ, ИК, Кафедра вычислительной техники

Введение

Циклические избыточные коды (CRC) в основном используют для определения ошибок при передаче данных по различным протоколам (Ethernet, ZigBee), а также при архивации данных (WinRaR, WinZip). Существуют разные алгоритмы вычисления CRC и способы их реализации. Некоторые программные реализации алгоритмов вычисления CRC (обратный табличный алгоритм), совместимых с WINRAR, PKZIP, ETHERNET можно встретить в свободном доступе [7]. Также в литературе [10] можно встретить другие реализации вычисления CRC, дающие результат, отличный от алгоритмов, совместимых с WINRAR, PKZIP, ETHERNET (прямой табличный алгоритм) [4,5]. Существуют и другие реализации CRC, не встречающиеся в свободном доступе, которые будут описаны в данной работе.

1. Программная реализация табличного алгоритма

Особенность данной реализации заключается в побайтовом вычислении контрольной суммы CRC32 с использованием таблицы предвычисленных значений на основе образующего полинома (104C11DB7h) [9,7]. Входным параметром в данном случае является имя файла, для которого рассчитывается контрольная сумма, а выходным – значение CRC32 в шестнадцатеричной системе счисления. Далее представлен пример реализации вычисления контрольной суммы CRC32, совместимой с WINRAR, PKZIP, ETHERNET на языке C++. Полный текст исходного кода приведён в [8].

```
unsigned long crc32_table [256] =
{
        0x00000000, 0x77073096, 0xEE0E612C, 0x990951BA,
        ……......
        0xB40BBE37, 0xC30C8EA1, 0x5A05DF1B, 0x2D02EF8D
}

int Get_CRC(char* text)
{
    unsigned long ulCRC = 0xffffffff;
    int len;
    unsigned char* buffer;

    len = strlen(text);
    buffer = (unsigned char*)text;
    while(len--)
        ulCRC = (ulCRC >> 8) ^ crc32_table[(ulCRC & 0xFF) ^ *buffer++];
    return ulCRC ^ 0xffffffff;
}
```

В приведённом исходном коде crc32_table – таблица предвычисленных значений (всего содержит 256 значений), len – размер сообщения в байтах, buffer –буфер, содержащий сообщение, для которого рассчитывается CRC32, ulCRC – контрольная сумма CRC32.

По исходным кодам программной реализации был восстановлен табличный алгоритм вычисления CRC32 (рис.1) и назван обратным табличным алгоритмом, так как данные считываются с младших байт [6].

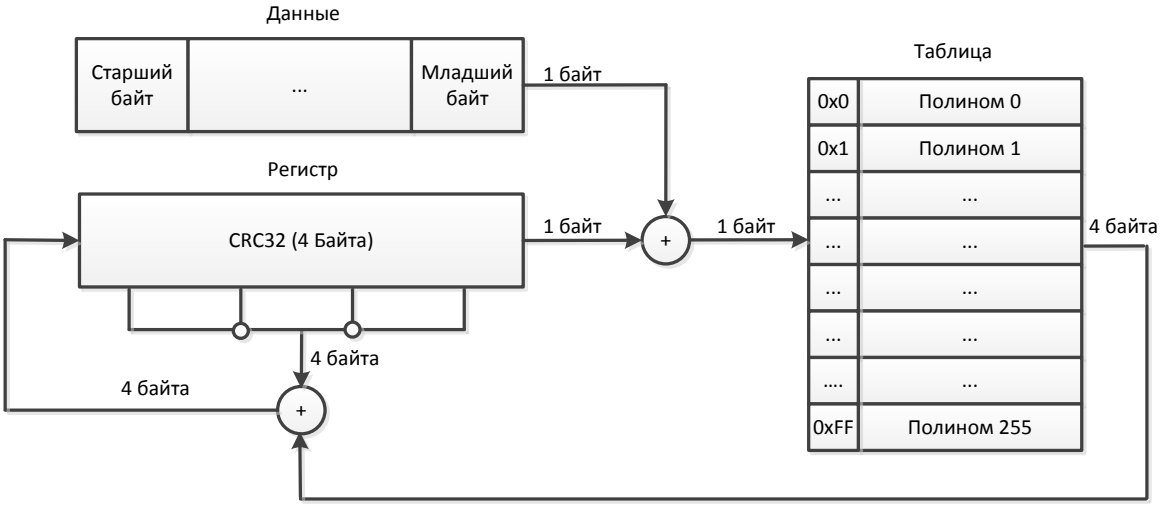

Рис. 1. *Схематичное описание обратного табличного алгоритма (разработано автором)*

2. Программная реализация матричного алгоритма

Реализация матричного алгоритма вычисления CRC32 заключается в применении операции умножения вектора (выдвинутый байт) на матрицу по модулю 2 [8,6]. Матрица рассчитывается на основе образующего полинома [9,4] и имеет меньшую размерность, чем при табличном алгоритме. На основе программной реализации табличного алгоритма был получен исходный код матричного алгоритма вычисления CRC32, пример которого представлен далее.

```
unsigned int crc32_table[8] =
    {
        0x77073096, 0xEE0E612C, 0x76DC419, 0xEDB8832,
      0x1DB71064,0x3B6E20C8,0x76DC4190, 0xEDB88320
     };



int Get_CRC(char* text)
{
    unsigned long ulCRC = 0xffffffff;
    int len;
```

```
    unsigned char* buffer;

    len = strlen(text);
    buffer = (unsigned char*)text;
    int i = 0;
    unsigned int crcb = 0;

    while(len--)
   {
        crcb = (ulCRC & 0xFF) ^ buffer[i++];
          ulCRC = (ulCRC >> 8) ^
          (crcb & 0x1) * crc32_table[0]  ^
          ((crcb & 0x2) >> 1) * crc32_table[1]  ^
        ((crcb & 0x4) >> 2) * crc32_table[2]  ^
          ((crcb & 0x8) >> 3) * crc32_table[3]  ^
        ((crcb & 0x10) >> 4) * crc32_table[4]  ^
          ((crcb & 0x20) >> 5) * crc32_table[5]  ^
        ((crcb & 0x40) >> 6) * crc32_table[6]  ^
          ((crcb & 0x80) >> 7) * crc32_table[7]  ;
    }
    return ulCRC ^ 0xffffffff;
}
```

В данном примере в функции Get_CRC был заменён основной цикл вычисления контрольной суммы согласно описанию алгоритма [4,5], а также изменена матрица crc32_table. На рис. 2 представлено схематичное описание матричного алгоритма согласно его программной реализации.

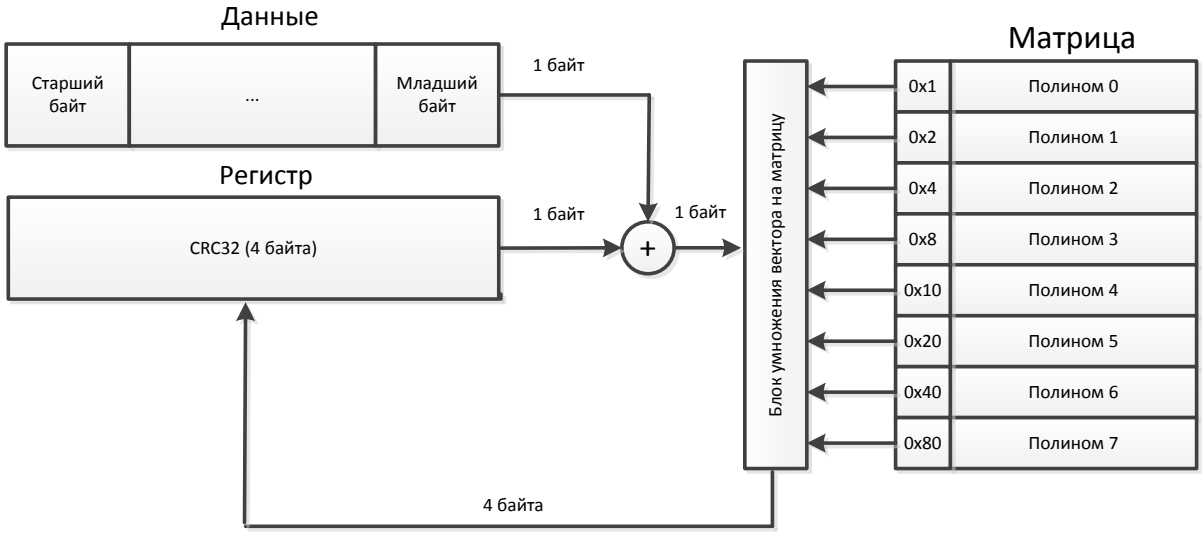

Рис. 2. *Схематичное описание матричного алгоритма (разработано автором)*

Особенность матричного алгоритма заключается в том, что его можно модифицировать, используя сдвиг по 2 или 4 байта. При этом будет увеличиваться размер матрицы, используемой при вычислениях. Для двухбайтового сдвига матрица расширится до 64-х байт; для четырёхбайтового сдвига – до 128 байт [4].

Далее приведен пример и описание реализации (рис.3) для двухбайтового матричного алгоритма.

```
unsigned int crc32_table[16] =
{
        0x191B3141,0x32366282,0x646CC504,0xC8D98A08,
        0x4AC21251,0x958424A2, 0xF0794F05,0x3B83984B,
        0x77073096, 0xEE0E612C, 0x76DC419, 0xEDB8832,
        0x1DB71064, 0x3B6E20C8, 0x76DC4190, 0xEDB88320
};
int Get_CRC(char* text)
{
    unsigned long ulCRC = 0xffffffff;
    int len,remind_len;

    int16_t* buffer;
    unsigned char* remind_buffer;

    len = strlen(text)/2;
```

```
buffer = (int16_t*)text;
remind_buffer = (unsigned char*)text;
int i = 0;
unsigned int crcb = 0;
remind_len = strlen(text)%2;

while(len--)
{

    crcb = (ulCRC& 0xFFFF) ^ buffer[i++];
      ulCRC= (ulCRC>> 16) ^
      (crcb & 0x1) * crc32_table[0]  ^
      ((crcb & 0x2) >> 1) * crc32_table[1]  ^
    ((crcb & 0x4) >> 2) * crc32_table[2]  ^
      ((crcb & 0x8) >> 3) * crc32_table[3]  ^
    ((crcb & 0x10) >> 4) * crc32_table[4] ^
      ((crcb & 0x20) >> 5) * crc32_table[5] ^
    ((crcb & 0x40) >> 6) * crc32_table[6] ^
      ((crcb & 0x80) >> 7) * crc32_table[7] ^
    ((crcb & 0x100)>>8) * crc32_table[8]  ^
      ((crcb & 0x200) >> 9) * crc32_table[9]    ^
    ((crcb & 0x400) >> 10) * crc32_table[10]  ^
      ((crcb & 0x800) >> 11) * crc32_table[11]  ^
    ((crcb & 0x1000) >> 12) * crc32_table[12] ^
      ((crcb & 0x2000) >> 13) * crc32_table[13] ^
    ((crcb & 0x4000) >> 14) * crc32_table[14] ^
      ((crcb & 0x8000) >> 15) * crc32_table[15]  ;
}

if ( remind_len != 0 )
{
    crcb = (ulCRC& 0xFF) ^ remind_buffer [strlen(text)-1];
      ulCRC= (ulCRC>> 8) ^
     (crcb & 0x1) * crc32_table[8]  ^
     ((crcb & 0x2) >> 1) * crc32_table[9]  ^
```

```
        ((crcb & 0x4) >> 2) * crc32_table[10]  ^
        ((crcb & 0x8) >> 3) * crc32_table[11]  ^
        ((crcb & 0x10) >> 4) * crc32_table[12] ^
        ((crcb & 0x20) >> 5) * crc32_table[13] ^
        ((crcb & 0x40) >> 6) * crc32_table[14] ^
        ((crcb & 0x80) >> 7) * crc32_table[15] ;
    }
    return ulCRC ^ 0xffffffff;
}
```

В приведённом примере увеличена размерность матрицы crc32_table, и изменён тип данных для буфера сообщения с однобайтового (char) на двухбайтовый (int16_t). Так как в данной реализации цикл обрабатывает сообщение по 2 байта, то для случая, если размер сообщения не кратен двум байтам, включен дополнительный побайтовый расчёт CRC32 (после основного цикла).

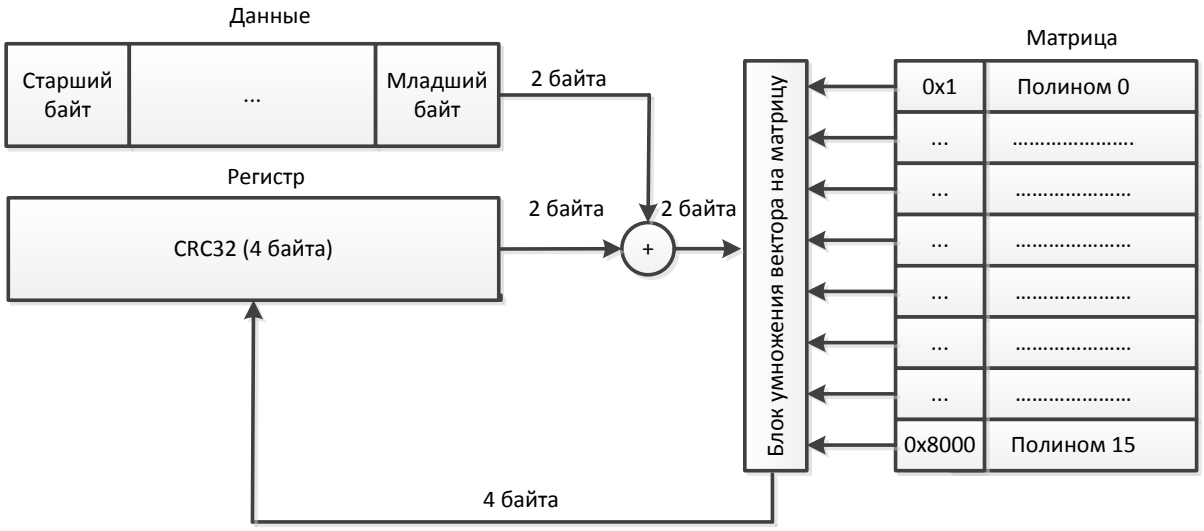

Рис. 3. *Схематичное описание матричного двухбайтового алгоритма (разработано автором)*

Для реализации четырёхбайтового матричного алгоритма необходимо внести соответствующие модификации, такие как увеличение размерности матрицы и обработка сообщения в цикле по 4 байта.

```
unsigned int crc32_table[32] =
{
        0xB8BC6765, 0xAA09c88b, x8F629757, 0xC5b428EF,
        0x5019579F, 0xA032AF3E, 0x9B14583D, 0xED59B63B,
        0x1C26A37, 0x384d46E, 0x709A8DC, 0xE1351B8,
        0x1C26A370, 0x384d46E0, 0x709A8DC0, 0xE1351B80,
```

```
        0x191B3141, 0x32366282, 0x646CC504, 0xC8D98A08,
        0x4AC21251, 0x958424A2, 0xF0794F05, 0x3B83984B,
        0x77073096, 0xEE0E612C, 0x76DC419, 0xEDB8832,
        0x1DB71064, 0x3B6E20C8, 0x76DC4190, 0xEDB88320
};
int Get_CRC(char* text)
{
    unsigned long ulCRC = 0xffffffff;
    int len,remind_len;

    int32_t* buffer;
    unsigned char* remind_buffer;

    len = strlen(text)/4;
    buffer = (int32_t*)text;
    remind_buffer = (unsigned char*)text;
    int i = 0;
    unsigned int crcb = 0;
    remind_len = strlen(text)%4;

    while(len--)
    {
        crcb = (ulCRC& 0xFFFFFFFF) ^ buffer[i++];

            ulCRC =
            (crcb & 0x1) * crc32_table[0]  ^
            ((crcb & 0x2) >> 1) * crc32_table[1]  ^
          ((crcb & 0x4) >> 2) * crc32_table[2]  ^
            ((crcb & 0x8) >> 3) * crc32_table[3]  ^
          ((crcb & 0x10) >> 4) * crc32_table[4] ^
            ((crcb & 0x20) >> 5) * crc32_table[5] ^
          ((crcb & 0x40) >> 6) * crc32_table[6] ^
            ((crcb & 0x80) >> 7) * crc32_table[7] ^
          ((crcb & 0x100) >> 8) * crc32_table[8] ^
            ((crcb & 0x200) >> 9) * crc32_table[9]    ^
```

```
  ((crcb & 0x400) >> 10) * crc32_table[10]  ^
   ((crcb & 0x800) >> 11) * crc32_table[11]  ^
  ((crcb & 0x1000) >> 12) * crc32_table[12] ^
   ((crcb & 0x2000) >> 13) * crc32_table[13] ^
  ((crcb & 0x4000) >> 14) * crc32_table[14] ^
   ((crcb & 0x8000) >> 15) * crc32_table[15] ^
  ((crcb & 0x10000) >> 16) * crc32_table[16] ^
   ((crcb & 0x20000) >> 17) * crc32_table[17] ^
  ((crcb & 0x40000) >> 18) * crc32_table[18]  ^
   ((crcb & 0x80000) >> 19) * crc32_table[19]  ^
  ((crcb & 0x100000) >> 20) * crc32_table[20] ^
   ((crcb & 0x200000) >> 21) * crc32_table[21] ^
  ((crcb & 0x400000) >> 22) * crc32_table[22] ^
   ((crcb & 0x800000) >> 23) * crc32_table[23] ^
  ((crcb & 0x1000000) >> 24) * crc32_table[24]  ^
   ((crcb & 0x2000000) >> 25) * crc32_table[25]  ^
  ((crcb & 0x4000000) >> 26) * crc32_table[26]  ^
   ((crcb & 0x8000000) >> 27) * crc32_table[27]  ^
  ((crcb & 0x10000000) >> 28) * crc32_table[28] ^
   ((crcb & 0x20000000) >> 29) * crc32_table[29] ^
  ((crcb & 0x40000000) >> 30) * crc32_table[30] ^
   ((crcb & 0x80000000) >> 31) * crc32_table[31] ;
}

if (  remind_len != 0 )
{

      while ( remind_len-- )
      {
        crcb = (ulCRC& 0xFF) ^ remind_buffer[strlen(text) - remind_len - 1];
          ulCRC= (ulCRC>> 8) ^
          (crcb & 0x1) * crc32_table[24]  ^
          ((crcb & 0x2) >> 1) * crc32_table[25]  ^
        ((crcb & 0x4) >> 2) * crc32_table[26]  ^
          ((crcb & 0x8) >> 3) * crc32_table[27]  ^
```

```
                ((crcb & 0x10) >> 4) * crc32_table[28] ^
                  ((crcb & 0x20) >> 5) * crc32_table[29] ^
                ((crcb & 0x40) >> 6) * crc32_table[30] ^
                  ((crcb & 0x80) >> 7) * crc32_table[31] ;
            }
        }

    return ulCRC ^ 0xffffffff;
}
```

В данном примере для расчёта CRC для сообщения, длина которого не кратна 4-м байтам, применяется дополнительный цикл с побайтовой обработкой. На рис. 4 приведено описание матричного четырёхбайтового алгоритма.

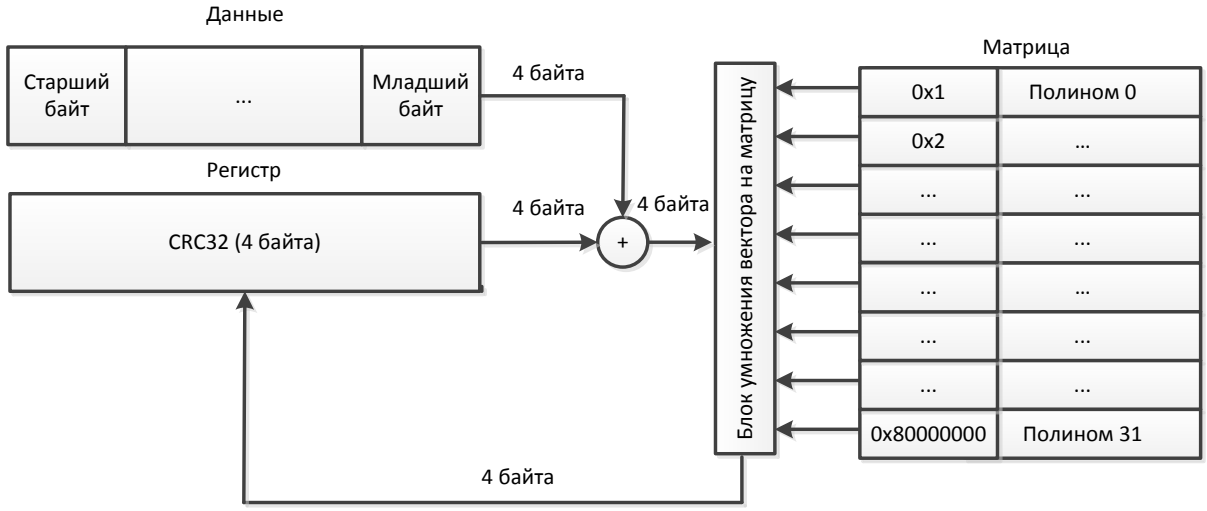

Рис. 4. *Схематичное описание матричного четырехбайтового алгоритма (разработано автором)*

Таким образом, для программной реализации матричного алгоритма вычисления контрольной суммы CRC32 в функции Get_CRC исходного кода [1] необходимо заменить основной цикл вычисления CRC на цикл, соответствующий описанию матричного алгоритма. Также необходимо изменить матрицу предвычисленных значений и учесть в коде обработку сообщений, не кратных требуемому количеству байт согласно реализуемому матричному алгоритму.

Основным преимуществом матричного алгоритма над табличным является то, что для его реализации требуется значительно меньше памяти, при этом алгоритм является достаточно простым для модификации. Так, для реализации табличного алгоритма требуется 1 Кб памяти на хранение таблицы, в то время, как для матричного алгоритма всего 32 байта, для матричного двухбайтового – 64 байта, для четырёхбайтового – 128 байт.

Заключение

В данной статье приведены примеры программных реализаций алгоритмов вычисления контрольной суммы CRC32. Используя примеры из данной работы и исходный код программы, имеющийся в свободном доступе, можно легко осуществить программную реализацию матричного алгоритма вычисления CRC32. Для рассмотренных программных реализаций восстановлены описания алгоритмов вычисления контрольной суммы. На основе приведённых примеров можно сделать вывод о том, что для реализации матричного алгоритма требуется меньше памяти, чем для табличного, при этом реализацию матричного алгоритма достаточно просто модифицировать, увеличив количество обрабатываемых байт за итерацию в основном цикле исходного кода вычисления CRC.

ЛИТЕРАТУРА

1. Буркатовская Ю.Б., Мальчуков А.Н., Осокин А.Н. Быстродействующие алгоритмы деления полиномов в арифметике по модулю два //Известия Томского политехнического университета, 2006. – т.309 – № 1. С. 19-24
2. Мальчуков А.Н., Осокин А.Н. Быстродействующие алгоритмы вычисления контрольной суммы на примере CRC8 // Молодежь и современные информационные технологии: Сборник трудов VIII Всерос. научно-практ. конф. студентов, аспирантов и молодых ученых. – Томск, 3-5 марта 2010. – Томск: СПБ Графикс, 2010. – С. 34–35.
3. Мальчуков А.Н., Осокин А.Н. Быстрое вычисление контрольной суммы CRC: таблица против матрицы //Прикладная информатика, 2010. – т. – № 2(26). – С. 58-63.
4. Мыцко Е. А. , Мальчуков А. Н. Особенности программной реализации вычисления контрольной суммы CRC32 на примере PKZIP, WINZIP, ETHERNET [Электронный ресурс] // Вестник науки Сибири. Серия: Информационные технологии и системы управления. – 2011 – №. 1 – С. 279-282. – Режим доступа: http://sjs.tpu.ru/journal/issue/view/2/showToc/sect/4
5. Мыцко Е.А., Мальчуков А.Н. Исследование программных реализаций табличного и матричного алгоритмов вычисления контрольной суммы CRC32 [Электронный ресурс] // Вестник науки Сибири. Серия: Информационные технологии и системы управления. – 2011 – №. 1 – С. 273-278. – URL: http://sjs.tpu.ru/journal/issue/view/2/ showToc/sect/4 (дата обращения 6.08.12)
6. Мыцко Е. А. , Мальчуков А. Н. Исследование программных реализаций алгоритмов вычисления CRC совместных с PKZIP, WINRAR, ETHERNET // Известия Томского политехнического университета. – 2013 – Т. 322 – №. 5. – С. 170-175
7. Темников Ф.Е., Афонин В.А., Дмитриев В.И. Теоретические основы информационной техники. – 2-е изд., испр. и доп. – М.: Энергия, 1979. – 512 с.
8. 32 bit Cyclic Redundancy Check Source Code for C++. // Create Window Website. 2011. URL: http://www.createwindow.com/programming/crc32/ (дата обращения: 01.04.2014).
9. Koopman P. 32-Bit Cyclic Redundancy Codes for Internet Applications // Intern. Conf. on Dependable Systems and Networks (DSN). – Washington, July 2002. – Washington, DC, 2002. – P. 459–468.
10. Ross N.W. A Painless Guide to CRC Error Detection Algorithms. // Dr Ross Williams. 1993. URL: http://www.ross.net/crc/download/crc_v3.txt (дата обращения: 01.04.2014).

Рецензент: Ким Валерий Львович, д. т. н., профессор кафедры вычислительной техники ИК ТПУ.

Evgeniy Mytsko
FSAEI HE National Research Tomsk polytechnic university
Russia, Tomsk
E-Mail: evgenrus70@mail.ru

Software implementation examples of algorithms for computing the cyclic redundancy check 32 checksum compatible with winrar, pkzip, Ethernet

Abstract. Some specific software implementations examples of algorithms for computing the checksum cyclic redundancy check 32 were considered in the paper. Based on the source code freely available, software implementations for a one-byte, two-byte and four-byte matrix computation algorithms of cyclic redundancy check 32 were implemented. Descriptions of checksum algorithms were restored for considered software implementations. Differences in the software implementation of the table and matrix computation algorithms of cyclic redundancy check 32 were examined. It was shown that for a software implementation of the matrix algorithm must replace the main loop of free source code as well as change the predicted values of the matrix. The practical significance of these examples software algorithms of cyclic redundancy check 32 checksum was shown. It was shown on the source code examples that the matrix algorithm is very easy to modify increasing the number of bytes processed per iteration, as well as its implementation requires significantly less memory. Thus, for the algorithm implementation it requires 1 KB of memory for storing a table, while the matrix algorithm for a total of 32 bytes, for a double-byte - 64 bytes for the four-byte - 128 bytes.

Keywords: checksum; software implementation; tabular algorithm; matrix algorithm cyclic redundancy check 32; source code; single-byte; double-byte; four-byte; data.

Identification number of article 140TVN314

REFERENCES

1. Burkatovskaja Ju.B., Mal'chukov A.N., Osokin A.N. Bystrodejstvujushhie algoritmy delenija polinomov v arifmetike po modulju dva //Izvestija Tomskogo politehnicheskogo universiteta, 2006. – t.309 – № 1. S. 19-24
2. Mal'chukov A.N., Osokin A.N. Bystrodejstvujushhie algoritmy vychislenija kontrol'noj summy na primere CRC8 // Molodezh' i sovremennye informacionnye tehnologii: Sbornik trudov VIII Vseros. nauchno-prakt. konf. studentov, aspirantov i molodyh uchenyh. – Tomsk, 3-5 marta 2010. – Tomsk: SPB Grafiks, 2010. – S. 34–35.
3. Mal'chukov A.N., Osokin A.N. Bystroe vychislenie kontrol'noj summy CRC: tablica protiv matricy //Prikladnaja informatika, 2010. – t. – № 2(26). – S. 58-63.
4. Mycko E. A. , Mal'chukov A. N. Osobennosti programmnoj realizacii vychislenija kontrol'noj summy CRC32 na primere PKZIP, WINZIP, ETHERNET [Jelektronnyj resurs] // Vestnik nauki Sibiri. Serija: Informacionnye tehnologii i sistemy upravlenija. – 2011 – №. 1 – C. 279-282. – Rezhim dostupa: http://sjs.tpu.ru/journal/issue/view/2/showToc/sect/4
5. Mycko E.A., Mal'chukov A.N. Issledovanie programmnyh realizacij tablichnogo i matrichnogo algoritmov vychislenija kontrol'noj summy CRC32 [Jelektronnyj resurs] // Vestnik nauki Sibiri. Serija: Informacionnye tehnologii i sistemy upravlenija. – 2011 – №. 1 – C. 273-278. – URL: http://sjs.tpu.ru/journal/issue/view/2/ showToc/sect/4 (data obrashhenija 6.08.12)
6. Mycko E. A. , Mal'chukov A. N. Issledovanie programmnyh realizacij algoritmov vychislenija CRC sovmestnyh s PKZIP, WINRAR, ETHERNET // Izvestija Tomskogo politehnicheskogo universiteta. – 2013 – T. 322 – №. 5. – C. 170-175
7. Temnikov F.E., Afonin V.A., Dmitriev V.I. Teoreticheskie osnovy informacionnoj tehniki. – 2-e izd., ispr. i dop. – M.: Jenergija, 1979. – 512 s.
8. 32 bit Cyclic Redundancy Check Source Code for C++. // Create Window Website. 2011. URL: http://www.createwindow.com/programming/crc32/ (data obrashhenija: 01.04.2014).
9. Koopman P. 32-Bit Cyclic Redundancy Codes for Internet Applications // Intern. Conf. on Dependable Systems and Networks (DSN). – Washington, July 2002. – Washington, DC, 2002. – P. 459–468.
10. Ross N.W. A Painless Guide to CRC Error Detection Algorithms. // Dr Ross Williams. 1993. URL: http://www.ross.net/crc/download/crc_v3.txt (data obrashhenija: 01.04.2014).