CRC Checksum Calculation

The following code for performing Cyclic Redundancy Check (CRC) checksums is provided in case a determination is made that the Internet Protocol and/or the TCP should use a CRC procedure.

: Polynomial CRC algorithm for PDP-10.
: Hacked for use in internet stuff by David P. Reed (DPReMIT-ML)

: computes standard CRC-16 checksum, remainder of message with
: polynomial $x^{16}+x^{15}+x^{2}+1$. Method used is generalization of
: Essentially, it is this.
: For 32 bit bytes, message is broken up into a sequence of bytes
: $M[i]$. The notation $m[i,j]$ is used for bits of byte $i$, where
: $m[i,0]$ is the first bit to be checksummed (stored in leftmost
: bit of byte).
: $U[i]$ is the upper 16 bits, expressed as a polynomial:
: $U[i] = \text{sum}(m[i,j] \times x^{(15-j)}, j=0,15)$
: $L[i]$ is the lower 16 bits, expressed similarly.
: $L[i] = \text{sum}(m[i,j] \times x^{(15-j)}, j=0,15)$
: So we can express $M$:
: $M[i] = U[i] \times x^{16} + L[i]$

: The input is the initial remainder polynomial $R(0)$, and compute the
: remainder of the polynomial:
: $R(0) \times x^{(32 \times N)} + \text{sum}(M[i] \times x^{(N-16-32 \times i)}, i=0,N-1)$
: when divided by the CRC-16 polynomial.
: This is done a 32-bit byte at a time, since the
: remainder after the $i$th byte can be expressed as:
: $R[i] = P[i] \times x^{(15 \times i)2+1} + W[i]$
: $R(N)$ is the desired message checksum. $P[i]$ is the parity of the
: first 32$i$ bits of the message as in the notation of Kirstein
: and Higginson.
: $W[i]$ is defined to be:

: $W[0] = \text{initial remainder on input}$
: $W[i+1] = \{(W[i]+U(i)) \times x^{(4+4 \times i)}+L[i] \times x^{(12+4 \times i)}
: + (A+B+C) \times x^{(4+15+1)}
: + A \times x^{15}
: + (A+C) \times x^{16}
: + (B+C) \times x^{4}
: + (A+C) \times x^{2}
: + (A+C) \times x\}$
: where $[u]$ stands for the remainder of $u$ when divided
: by $x^{16}$ (truncating terms of order higher than 16), and given
: that $w[i,j]$ is the coefficient of $x^{(15-j)}$ in $W[i]$,;
: $A = w[i,0] \times m[i,0]$
CRC Checksum Calculation

B = w[i,1]+m[i,1]
C = A + w[i,2]+m[i,2]+m[i,16]
D = B + w[i,3]+m[i,3]+m[i,18]+m[i,17]

The speed of the algorithm comes from the fact that by cleverly

doing the multiplications of the terms in the i's, A, B, C,

and D are generated as coefficients of the terms to be truncated
by the i's.

register definitions:

inptr=10 ; byte pointer input to crc routines.
bytecnt=11 ; byte count input to crc32 routine. (32-bit
bytes).
parity=4 ; parity accumulator for CRC, message parity
 crct=parity+1 ; for crc, must be adjacent to rem and parity
 rem=crct+1 ; high 16 bits of rem are CRC remainder (i/o)
t=7
p=17
tyi=1
tyo=2

Usage: to get crc for a message, first call crcinit.

Then, make a sequence of calls to crc32, crc16, and crc8, in the

order the message bits are to be checksummed.crc32 does a

sequence of 32-bit bytes, while crc16 and crc8 do single 16 and 8

bit bytes. parity and rem are registers that must be preserved

across multiple calls. each crc routine takes a byte pointer as

input, incrementing it (once for crc8 and crc16, and at least once

for crc32). crc32 takes a byte (word) count, as well.

the crc is finished by calling crcfin.

when the crc is done, rem contains the crc in its high-order 16

bits, and possibly some random bits in the low order 20.

crcinit: setz parity,
    hrlzi rem,-4 ; clear parity accumulator.
    ; initial remainder is
    ; x15+x14+. . . +x+1
    popj p,

crc on 32 bit bytes. fastest of the three CRC's.

crc32: ildb crct,inptr ; get next word of input (right 4
    ; zero).
    lsh crct,36,-32 ; get to left end. This and
    ; prev could be optimized to
11 September 1978
IEN: 56

CRC Checksum Calculation

xor parity, crct
xor crct, rem
lsbc crct, 16.-36.
lsb rem, 16.-36.
move t, crct
lsb crct, 2
xor crct, t
xor crct, rem
lsb crct, 1
xor crct, rem
lsbc crct, 1-16.

xor rem, crctb (crct)
sogl bytcnt, crc32
popj p,

; a move off an aoji counter.
; accumulate parity
; xor in 16-bit remainder-so-far
; high 16 bits in crct, low in rem
; and get low bits in low 16 bits.
; copy high 16 bits.
; multiply by x12
; and xor in.
; xor in low 16 bits.
; multiply by x
; and xor in lower 16 bits again.
; and multiply by x, then shift so in
; proper place in rem. crct then has
; 4 bits shifted out in its low order
; bits, and correctly insert these 4
; bits, count down bytes remaining

; crc16 does one 16-bit byte.

crc16: ildb crct, inptr
xor parity, crct
lsb rem, 16.-36.
xorb crct, rem
lsb crct, 1
xor crct, rem
lsbc crct, 1-16.
xor rem, crctb (crct)
popj p,

; get 16-bit byte.
; get to right end.
; xor with rem so far.
; xor in rem.
; and lsb again, then move to final
; reset, fix up rem (only first four
; entries used) and return.

; crc8 does one 8-bit byte.

crc8: ildb t, inptr
setz crct,
lsbc crct, 8.
xor crct, t
xor parity, crct

lsb crct, 36.-16+1.
xor rem, crct
lsb crct, 1
xor rem, crct
popj p,

; get 8-bit byte.
; move low order byte of remainder to
; high byte. add in new byte
; parity := parity xor new byte xor
; high byte of W
; shift to low order byte of high
; 16 bits, mult by x
; and add to rem
; and mult by x
; and add again to rem.
; crcfin finishes up a sequence of 16-bit and 32-bit CRC calls.

crcfin: move crct,parity
rot parity,18.
xorb parity,crct
rot parity,9.
xor parity,crct
and parity,[042104210421]

idivi parity,17
trne crct,1
xor rem,[100803+2b]
popj p,

ccrctb: 0+20,+0-21,+0-22,+0-23,+0-24,+0-25.
100001+20,+0-21,+1+22,+0-23,+0-24,+0-25.
100001+20,+1+21,+1+22,+1+23,+0-24,+0-25.
0-20,+1+21,+1+22,+1+23,+1+24,+0-25.
100001+20,+1+21,+0-22,+1+23,+1+24,+0-25.
0-20,+1+21,+1+22,+1+23,+1+24,+0-25.
0-20,+0-21,+1+22,+1+23,+1+24,+0-25.
100001+20,+0-21,+0-22,+0-23,+1+24,+0-25.
100001+20,+1+21,+0-22,+0-23,+1+24,+0-25.
0-20,+1+21,+1+22,+0-23,+1+24,+1+25.
0-20,+0-21,+1+22,+0-23,+1+24,+1+25.
0-20,+0-21,+1+22,+1+23,+1+24,+1+25.
100001+20,+0-21,+0-22,+1+23,+1+24,+1+25.
0-20,+0-21,+0-22,+1+23,+0-24,+1+25.
100001+20,+0-21,+1+22,+0-23,+0-24,+1+25.
100001+20,+1+21,+1+22,+0-23,+0-24,+1+25.
0-20,+1+21,+0-22,+0-23,+0-24,+1+25.
; testing procedure -- runs a diagnostic check of the three routines,
; then times it.

go:           move p,[#1000,stack-1] ; initialize
            .open ttyi,[8,,'tty']
            .lose 1000
            .open ttyo,[1,,'tty']
            .lose 1000

        crc100=100857 ; best by test!

            pushj p,crcinit
            movei bytecnt,25, ; do 25. words of zeros 32 bits at
            move inptr,[444000,,zeros] ; a time.
            pushj p,crc32
            pushj p,crcfin
            lsh rem,16,-36.
            caie rem,crc100 ; compare with correct crc of
            .value ; 800 zeros.

            pushj p,crcinit ; do 25. words 16 bits at a time,
            movei bytecnt,25,2 ; for a check
            move inptr,[442000,,zeros]
            pushj p,crc16
            sojg bytecnt,,-1
            pushj p,crcfin
            lsh rem,16,-36.
            caie rem,crc100 ; compare with correct crc of
            .value ; 800 zeros.

            pushj p,crcinit ; do 25 words 8 bits at a time for
            movei bytecnt,25,4 ; a check
            move inptr,[441000,,zeros]
            pushj p,crc8
            sojg bytecnt,,-1
            pushj p,crcfin
            lsh rem,16,-36.
            caie rem,crc100 ; compare with correct crc of
            .value ; 800 zeros.
; timing of a checksum applied to a 1024 octet message.
;a-1
    movei a,10.
    movem a,trycount

trylp:
; start timing.
    .suset [.rrunt,.strtim] ; read starting runtime
;    .call klpfs
;    .lose 1000

; set byte pointer to beginning of internet header.
    move inptr,[444000,.inhdr]

; do 31. words. and then do one 16. bit word.

    movei bytecnt,31.
    pushj p,crc32
; now do 1 odd 16 bit byte left at end.
    hrl inptr,002000 ; patch byte ptr to point to
    pushj p,crc16 ; next 16 bit byte.
    pushj p,crcfin ; finish up crc.

; finish timing
    .suset [.rrunt,.fintim] ; read final runtime
;    .call klpfs
;    .lose 1000
    move a,fintim ; compute runtime
    sub a,strtim
    camg a,mintime ; adjust mintime
    movem a,mintime
    sosl trycount
    jrst trylp

; type out results, timing statistics

    movoi a,[asciz /Min time: /]
    pushj p,typcout

    move a,mintime
ash a,2 : runtimes are in 4microsecond
; units
ash a,-12. ; runtimes are in units of 2**(12
on mc
(only)
subi a,448. ; 448. is magic correction for ml
(only)
subi a,210. ; 210. is magic constant for mc
(only)
pushj p,decpnt
movei a,[asciz / microseconds.]
pushj p,typeout
pushj p,terpri
.value [asciz :kill/]
inhdr: 21000001200 ; typical?
525250000000
00200000000
002203000000
002030000020
; following random code is "body" of message.
; block 28.
d=10
el=11
f=12
typeout:move f,a
for f,[446700,0]
typlp: ildb d,f
skipn d
popj p,
.iot tyo,d
jrst typlp
ding: .iot tyo,[7]
terpri: .iot tyo,[15]
.iot tyo,[12]
popj p,
.decpnt: push p,d
move d,a
pushj p,decpnl
pop p,d
popj p,
.decpnl: idivi d,12
push p,e
skipe d
pushj p,decpnl
pop p,d

Postel
addi d, 60
.io tyo, d
.popj p,

klpfs: setz
s.b 1 /klperf/
movei -4
move paeud
movem prevjob
movem prevpae
movem tbl
movem strtime
movem pel
setzm pe2

klpff: setz
s.b 1 /klperf/
movei -4
move paeud
movem prevjob
movem prevpae
movem tbl
movem fintime
movem pel
setzm pe2

tbl: 0
pel: 0
pe2: 0
prevpae: 0
prevjob: 0
paeud: 0

mintime: 377777777777
trycount: 0
strtime: 0
fintime: 0
zeros: block 25.

stack: block 1000
end go
;Local modes:
;Mode: midas
;Turn On Auto Save Mode: 1
;End:

----------
/ Subroutine for doing Internet CRC's with the IBM polynomial
/ CRC = X^16 + X^15 + X^12 + 1. The algorithm is adapted from
/ Higgins and Kirstein.
/
/
/ This version takes x memory references (max) and y instructions
/ (max) per z bit word. Typical timings are a usec per word on an
/ 11/70 with a cache and b usec on an 11/40 with 600 nsec MOS memory.
/
/
/ Written by D. Reed with assistance from N Chiappa.
/ MIT-LCS-CSRD 21/8/78
/
/
/ This version works for those of you who have a real operating
/ system (UNIX) on your machine with C. Others will have to mung
/ the program to use your calling conventions (and assembler).
/
/ For those who are puzzled, "S" = " ", "!" = "bitwise not",
/ and labels of the form "x f" and "x b" refer to the first "x"
/ forward or back from here.
/
/ C call is of form:
/ char wbuf;
/ int len;
/ struct
/ {
/   unsigned checksum;
/     unsigned parity;
/ }
/ chkres;
/
/ crc-strt(&chkres);
/ while (data-left()) crc(buf, len, &chkres);
/ crc-end(&chkres);
.globl +crc
+crc:
    mov     sp, r8        / Save arg pointer
    mov     r2, -(sp)     / Stash reg
    mov     r3, -(sp)     / Stash reg
    mov     r4, -(sp)     / Stash reg
    mov     r5, -(sp)     / Stash reg
    tst     (r8)+        / Go look at arg list
    mov     (r8)+, r2    / Data pointer
    mov     (r8)+, r3    / Size
    mov     s(r8), r4    / Return area pointer
    mov     r4, -(sp)     / Save pointer to return area
    jsr     pc, 1f        / Call into crc routine
    mov     (sp)+, r8    / Pick up pointer
    mov     r1, 2(r8)     / Return new par
    mov     r5, s(r8)     / New checksum
    mov     (sp)+, r5     / Restore regs and return
    mov     (sp)+, r4
    mov     (sp)+, r3
    mov     (sp)+, r2
    rts     pc

/ Here is where real CRC calculation starts

1:
    mov     (r4)+, r5    / Checksum so far
    mov     sr4, r1      / Parity so far
    bit     $1, r2       / See if odd byte
    bcle     1f
    jsr     pc, 3f       / Do the byte
    dec     r3           / Dec no of bytes and see if
    bne     1f           / any more
    rts     pc           / Only one byte
1:  
  asr  r3
  bcc  1f

  mov  $3f,  -(sp)  /* Do the odd byte at the end */

1:  
  asr  r3
  bcc  1f

  mov  (r2)+,  r0  /* Hack for jumping into */
  subb  r0     /* middle of loop */
  xor   r0,   r1
  xor   r0,   r5  /* Add in second 16 bits */
  mov   r5,   r0

  inc   r3
  clr   r4
  br    2f

1:  
  mov  (r2)+,  r0  /* Suck up next word */
  subb  r0     /* Dumb pdp11 byte numbering */
  xor   r0,   r1
  xor   r0,   r5
  mov   r5,   r0
  ext   r4  /* Initialize r4 with bit A */
  /* of 32 bit quan */
  aol   r5  /* Multiply by X*2 */
  aol   r5
  rol   r4  /* Shift in bit B */
  xor   r0,   r5  /* Done with first word */

  mov  (r2)+,  r0
  subb  r0
  xor   r0,   r1
  xor   r0,   r5  /* Add in second 16 bits */

2:  
  aol   r5  /* Multiply by X */
  rol   r4  /* Get bit C */
  xor   r0,   r5  /* Add in again */
  aol   r5  /* Multiply by X */
  rol   r4  /* Get bit D */

  aol   r4  /* Multiply by 2 for */
  /* table look up */
  mov   ctb(r4),r0  /* Table contains correction */
  /* for A,B,C & D */
  xor   r0,   r5
sob  r3,  1b
rts  pc

3:
  movb  (r2)+,  r0  / Do one byte
  subb  r5
  xor  r5,  r0
  bic  $377,  r8
  xor  r0,  r1  / Xor into parity
  bic  $377,  r5
  mov  r0,  r4
  asl  r0
  xor  r4,  r0
  asl  r0
  xor  r0,  r5
rts  pc  / End of CRC

.globl  +crc-strt
+crc-strt:  / You can do this in the program
           / if you want
  mov  sp,  r0  / Get to arg
  tst  (r0)+
  mov  *r0,  r0
  mov  $-1,  (r0)+  / Set initial checksum
  clr  *r0  / Set initial parity
rts  pc

.globl  +crc-end
+crc-end:
  mov  sp,  r0  / Get to arg
  tst  (r0)+
  mov  *r0,  r0
  mov  r2,  -(sp)  / Stash reg
  mov  2(r0),  r1  / Compute parity of bits in r1
  mov  r1,  r2
swab r1
xor r1, r2
mov r2, r1
asl r1
asl r1
asl r1
xor r1, r2
sxt r1
asl r2
asl r2
adc r1
asl r2
adc r1
adc r1
ror r1
bcs 1f
mov $100003, r1
xor r1, w8  / Xor into checksum
1: mov (sp)+, r2  / Restore reg
rts pc

crc-tb: 100003
66
74
100071
50
100055
100047
42

cbb: 0  / Note that offset into table may
        / be neg from here
100005
100017
12
100033
36
24
100021
Note: If you want to copy this code for testing on your machine, you might prefer the copy in the file <INTERNET-NOTEBOOK>_CRC-CODE.TXT at ISIE.