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 (DRReed-MIT-ML)

; 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] = sum(m[i,j]*x^(15-j), j=0,15)
; L[i] is the lower 16 bits, expressed similarly.
; L[i] = sum(m[i,j]*x^(15-j), j=0,15)
; So we can express M:
; M[i] = U[i]*x^16+L[i]

; The input is the initial remainder polynomial R(0), and compute the remainder of the polynomial:
; R(0)*x^i*(32*x) + sum(M[i]*x*(N-16-32*X)), 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]*x*(15+x)*2+i+U[i]
; R(N) is the desired message checksum. P[i] is the parity of the first 32*x[i] bits of the message as in the notation of Kirstein
; and Higginson.
; W[i] is defined to be:
; W[0] = initial remainder on input.
; W[i+1] = (W[i]+U[i])*(x^14+x^2)+L[i]*x^12+x)
; + (A+B+C)*x^15
; + (A+D) * x^16
; + (B+C) * x^13
; + (C+D) * x^9
; + (A+B+C) * x
; where W[i] stands for the remainder of W 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]*w[i,0]
The speed of the algorithm comes from the fact that by cleverly doing the multiplications of the terms in the \( A \)'s, \( B \), \( C \), and \( D \) are generated as coefficients of the terms to be truncated by the \( H \)'s.

Register definitions:

\[
\begin{align*}
\text{inptr} &= 10; & \text{byte pointer input to CRC routines.} \\
\text{bytecnt} &= 11; & \text{byte count input to CRC32 routine. (32-bit bytes)} \\
\text{parity} &= 4; & \text{parity accumulator for CRC, message parity} \\
\text{crcct} &= 4; & \text{for crc, must be adjacent to rem and parity} \\
\text{rem} &= \text{crcct} + 1; & \text{high 16 bits of rem are CRC remainder (I/O)} \\
\text{t} &= 7; & \text{for crc32} \\
\text{tyi} &= 2; & \text{for crc32} \\
\text{tyo} &= 2; & \text{for crc32}
\end{align*}
\]

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; & clear parity accumulator. 
hrzi rem, -4; & initial remainder is 
x115 + x14 + ... + x1
popj p,

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

crc32: ildb crcct, inptr; & get next word of input (right 4 bits)
zerocrcc; & zero.
Isb crcc, 36, -32; & get to left enc. This and prev could be optimized to
11 September 1978
IEN: 56

CRC Checksum Calculation

```assembly
xor parity, crct
xor crct, rem
lshc crct, 16, -36.
lsrem, 16, -36.
mov t, crct
lsh crct, 2
xor crct, t
xor crct, rem
lsh crct, 1
xor crct, rem
lshc crct, 1-16.

xor rem, crcrb(crcnt)
sojg bytecnt, 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 x^12
; and xor in.
; xor in low 16 bits.
; multiply by x
; and xor in low 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.

```assembly
crc16: ildb crct, inptr
xor parity, crct
lsh rem, 16, -36.
xor crct, rem
lsh crct, 1
xor crct, rem
lshc crct, 1-16.
xor rem, crcrb(crcnt)
popj p,
```

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

; crc8 does one 8 bit byte.

```assembly
crc8: ildb t, inptr
setz crct,
lshc crct, 8.
xor crct, t
xor parity, crct
lsh crct, 36, -16, +1.
xor rem, crct
lsh 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 crcr, parity
       rot parity, 18.
       xor parity, crcr ; now get parity of message bits.
       rot parity, 9. ; do it by first getting the two
       xor parity, crcr ; halves xored
       parity, [042104210421] ; upper 18 bits = lower 18 bits of
       and parity, [042104210421] ; both parity and crcr.
       every fourth bit (hack
       from hakmem)
       idivi parity, 17 ; now four 9 bit bytes are equal,
       xor rem, [100000-20] ; and parity of
       parity=1 (crcr) = number of bits
       on in any byte.
       test parity of message.
       fix rem based on parity.
       popj p,

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.+1+21.+1+22.+0-23.+1+24.+1+25.
0-20.+0-21.+1+22.+1+23.+1+24.+1+25.
11 September 1978
IEN: 56

CRC Checksum Calculation

; testing procedure -- runs a diagnostic check of the three routines, then times it.

go:       move p,[1-1000, stack-1] ; initialize
   .open tyi,[0, 'tty]
   .lose 1000
   .open tyo,[1, 'tty]
   .lose 1000

crc100=100057 ; best by test!

   pushj p, crcinit
   movei bytecrt, 25. ; do 25. words of zeros 32 bits at
   move intr, [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, for a check
   movei bytecrt, 25, x2
   move intr, [442000, zeros]
   pushj p, crc16
   sojg bytecrt, -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 a check
   movei bytecrt, 25, x4
   move intr, [441000, zeros]
   pushj p, crc8
   sojg bytecrt, -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 1023 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.
    hrli 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 4 microsecond
; runtimes are in units of 2^12
on mc
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: 210000001200 ; typical?
525250000000
002000000000
002030000000
002000000020
; following random code is "body" of message.
block 28.
d-10
e-11
f-12
typeout: move f, a
for f, [448700, 0]
typ1p: ildb d, f
skipn d
popj p,
.iot tyo, d
jrst typ1p
ding: .iot tyo, [7]
terpri: .iot tyo, [15]
n.iot tyo, [12]
.popj p,
.decpnt: push p, d
move d, a
pushj p, decpnl
.popj p,
.popj p,
.decpnl: idivi d, 12
.push p, e
.skip e d
.pushj p, decpnl
.popj p,
addi d, 60
.jot tye, d
popj p,

klpfs:
setz
sixbit /k1perf/
movci -4
movem pacud
movem prevjob
movem prevpae
movem tbl
movem strftime
movem pel
setzm pe2

klpff:
setz
sixbit /k1perf/
movci -4
movem pacud
movem prevjob
movem prevpae
movem tbl
movem fintime
movem pel
setzm pe2

tbl: 0
pel: 0
pe2: 0
prevpac: 0
prevjob: 0
pacud: 0

mintime: 3777777777777
trycount: 0
strftime: 0
ftime: 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
/ Higginson 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 21/8/76
/
/
/ This version works for those of you who have a real operating
/ system (UNIX) on your machine with C. Others will have to munge
/ the program to use your calling conventions (and assembler).
/
/ For those who are puzzled, "&" = ", "!, = "bitwise not",
/ and labels of the form "xf" and "xb" refer to the first "x"
/ forward or back from here.
/
/
/ C call is of form:
/
/ char xbuf;
/ int len;
/ struct unsigned checksum;
/ {
/   unsigned parity;
/ }
/ chk+res;
/
/ crc-strt(&chk+res);
/ while (data-left()) crc(buf, len,&chk+res);
/ crc-end(&chk+res);
.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   @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,   @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   @r4,   r1        / Parity so far
     bit   $1,   r2         / See if odd byte
     bcq   1f
     jsr   pc,   3f         / Do the byte
     dec   r3
     bne   1f
     rts   pc        / Only one byte
1:    asr    r3
     bcc    1f
     mov    $3f, -(sp)  ; Do the odd byte at the end
     mov    (r2)+, r0  ; Hack for jumping into
     suab   r0, r1    ; middle of loop
     xor    r0, r5    ; Add in second 16 bits
     xor    r0, r5
     mov    r5, r0
     inc    r3
     clr    r4
     br    2f
1:    mov    (r2)+, r0  ; Suck up next word
     suab   r0, r1    ; Dumb push byte numbering
     xor    r0, r5
     xor    r0, r5
     mov    r5, r0
     ext    r4
     aal    r5        ; Initialize r4 with bit A
     aal    r5        ; of 32 bit quant
     rol    r4        ; Multiply by X*2
     xor    r0, r5    ; Shift in bit B
     xor    r0, r5
     mov    (r2)+, r0
     suab   r0, r1
     xor    r0, r5
     xor    r0, r5
     asl    r5        ; Add in second 16 bits
     rol    r4        ; Multiply by X
     rol    r4        ; Get bit C
     xor    r0, r5    ; Add in again
     asl    r5        ; Multiply by X
     rol    r4        ; Get bit D
     asl    r4        ; Multiply by 2 for
      mov    ctb(r4),r5 ; Table contains correction
     xor    r0, r5
CRC Checksum Calculation

```
sub r3, 1
rts pc

3:  movb (r2)+, r0  / Do one byte
    swab r5
    xor r5, r0
    bic $377, r8  / Xor into parity
    xor r0, r1
    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)+, r0
    mov sr0, r0
    mov $-1, (r0)+  / Set initial checksum
    clr sr0  / Set initial parity

rts pc
```

```
.globl +crc-end
+crc-end:
    mov sp, r0  / Get to arg
    tst (r0)+, r0
    mov sr0, r0
    mov r2, -(sp)  / Stash reg
    mov 2(r0), r1  / Compute parity of bits in r1
    mov r1, r2
```

Postel
suab r1
xor r1, r2
mov r2, r1
asl r1
asl r1
asl r1
xor r1, r2
ext r1
asl r2
asl r2
adc r1
asl r2
adc r1
adc r1
ror r1
bcs 1f
mov $100003,r1
xor r1, r0 / Xor into checksum
1: mov (sp)+, r2 / Restore reg
rts pc

crc+tb: 100063
66
74
100071
50
100055
100047
42

csb: 0 / Note that offset into table may
be neg from here
100005
100017
12
100033
36
24
100021
11 September 1978

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.