```
#|

 Copyright (C) 1994 by Yuan Yu.  All Rights Reserved.

 This script is hereby placed in the public domain, and therefore unlimited
 editing and redistribution is permitted.

 NO WARRANTY

 Yuan Yu PROVIDES ABSOLUTELY NO WARRANTY.  THE EVENT SCRIPT IS PROVIDED "AS IS"
 WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT
 LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
 SCRIPT IS WITH YOU.  SHOULD THE SCRIPT PROVE DEFECTIVE, YOU ASSUME THE COST OF
 ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

 IN NO EVENT WILL Yuan Yu BE LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS,
 LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT
 OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO LOSS
 OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD
 PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH DAMAGES, OR
 FOR ANY CLAIM BY ANY OTHER PARTY.

|#
```

EVENT: Start with the library "memmove" using the compiled version.

```
;          Proof of the Correctness of the MEMCPY Function
#|
This is part of our effort to verify the Berkeley string library.  The
Berkeley string library is widely used as part of the Berkeley Unix OS.

This is the source code of memcpy function in the Berkeley string library.

typedef int word;               /* "word" used for optimal copy speed */

#define wsize   sizeof(word)
#define wmask   (wsize - 1)

/*
 * Copy a block of memory, handling overlap.
 * This is the routine that actually implements
 * (the portable versions of) bcopy, memcpy, and memmove.
```

```
 */
void *
memcpy(dst0, src0, length)
        void *dst0;
        const void *src0;
        register size_t length;
{
        register char *dst = dst0;
        register const char *src = src0;
        register size_t t;

        if (length == 0 || dst == src)          /* nothing to do */
                goto done;

        /*
         * Macros: loop-t-times; and loop-t-times, t>0
         */
#define TLOOP(s) if (t) TLOOP1(s)
#define TLOOP1(s) do { s; } while (--t)

        if ((unsigned long)dst < (unsigned long)src) {
                /*
                 * Copy forward.
                 */
                t = (int)src;   /* only need low bits */
                if ((t | (int)dst) & wmask) {
                        /*
                         * Try to align operands.  This cannot be done
                         * unless the low bits match.
                         */
                        if ((t ^ (int)dst) & wmask || length < wsize)
                                t = length;
                        else
                                t = wsize - (t & wmask);
                        length -= t;
                        TLOOP1(*dst++ = *src++);
                }
                /*
                 * Copy whole words, then mop up any trailing bytes.
                 */
                t = length / wsize;
                TLOOP(*(word *)dst = *(word *)src; src += wsize; dst += wsize);
                t = length & wmask;
                TLOOP(*dst++ = *src++);
```

```
        } else {
                /*
                 * Copy backwards.  Otherwise essentially the same.
                 * Alignment works as before, except that it takes
                 * (t&wmask) bytes to align, not wsize-(t&wmask).
                 */
                src += length;
                dst += length;
                t = (int)src;
                if ((t | (int)dst) & wmask) {
                        if ((t ^ (int)dst) & wmask || length <= wsize)
                                t = length;
                        else
                                t &= wmask;
                        length -= t;
                        TLOOP1(*--dst = *--src);
                }
                t = length / wsize;
                TLOOP(src -= wsize; dst -= wsize; *(word *)dst = *(word *)src);
                t = length & wmask;
                TLOOP(*--dst = *--src);
        }
done:
        return (dst0);
}
```

The MC68020 assembly code of the C function memcpy on SUN-3 is given as follows. This binary is generated by "gcc -O".

```
0x2610 <memcpy>:        linkw fp,#0
0x2614 <memcpy+4>:      moveml d2-d4,sp@-
0x2618 <memcpy+8>:      movel fp@(8),d3
0x261c <memcpy+12>:     movel fp@(16),d2
0x2620 <memcpy+16>:     moveal d3,a1
0x2622 <memcpy+18>:     moveal fp@(12),a0
0x2626 <memcpy+22>:     beq 0x26c4 <memcpy+180>
0x262a <memcpy+26>:     cmpal d3,a0
0x262c <memcpy+28>:     beq 0x26c4 <memcpy+180>
0x2630 <memcpy+32>:     bls 0x267c <memcpy+108>
0x2632 <memcpy+34>:     movel a0,d1
0x2634 <memcpy+36>:     movel d1,d0
0x2636 <memcpy+38>:     orl d3,d0
0x2638 <memcpy+40>:     movel #3,d4
0x263a <memcpy+42>:     andl d4,d0
```

```
0x263c <memcpy+44>:     beq 0x2662 <memcpy+82>
0x263e <memcpy+46>:     movel d1,d0
0x2640 <memcpy+48>:     eorl d3,d0
0x2642 <memcpy+50>:     movel #3,d4
0x2644 <memcpy+52>:     andl d4,d0
0x2646 <memcpy+54>:     bne 0x264e <memcpy+62>
0x2648 <memcpy+56>:     movel #3,d4
0x264a <memcpy+58>:     cmpl d2,d4
0x264c <memcpy+60>:     bcs 0x2652 <memcpy+66>
0x264e <memcpy+62>:     movel d2,d1
0x2650 <memcpy+64>:     bra 0x265a <memcpy+74>
0x2652 <memcpy+66>:     movel #3,d0
0x2654 <memcpy+68>:     andl d1,d0
0x2656 <memcpy+70>:     movel #4,d1
0x2658 <memcpy+72>:     subl d0,d1
0x265a <memcpy+74>:     subl d1,d2
0x265c <memcpy+76>:     moveb a0@+,a1@+
0x265e <memcpy+78>:     subl #1,d1
0x2660 <memcpy+80>:     bne 0x265c <memcpy+76>
0x2662 <memcpy+82>:     movel d2,d1
0x2664 <memcpy+84>:     lsrl #2,d1
0x2666 <memcpy+86>:     beq 0x266e <memcpy+94>
0x2668 <memcpy+88>:     movel a0@+,a1@+
0x266a <memcpy+90>:     subl #1,d1
0x266c <memcpy+92>:     bne 0x2668 <memcpy+88>
0x266e <memcpy+94>:     movel #3,d1
0x2670 <memcpy+96>:     andl d2,d1
0x2672 <memcpy+98>:     beq 0x26c4 <memcpy+180>
0x2674 <memcpy+100>:    moveb a0@+,a1@+
0x2676 <memcpy+102>:    subl #1,d1
0x2678 <memcpy+104>:    bne 0x2674 <memcpy+100>
0x267a <memcpy+106>:    bra 0x26c4 <memcpy+180>
0x267c <memcpy+108>:    addal d2,a0
0x267e <memcpy+110>:    addal d2,a1
0x2680 <memcpy+112>:    movel a0,d1
0x2682 <memcpy+114>:    movel a1,d0
0x2684 <memcpy+116>:    orl d1,d0
0x2686 <memcpy+118>:    movel #3,d4
0x2688 <memcpy+120>:    andl d4,d0
0x268a <memcpy+122>:    beq 0x26ac <memcpy+156>
0x268c <memcpy+124>:    movel a1,d0
0x268e <memcpy+126>:    eorl d1,d0
0x2690 <memcpy+128>:    movel #3,d4
0x2692 <memcpy+130>:    andl d4,d0
```

```
0x2694 <memcpy+132>:    bne 0x269c <memcpy+140>
0x2696 <memcpy+134>:    movel #4,d4
0x2698 <memcpy+136>:    cmpl d2,d4
0x269a <memcpy+138>:    bcs 0x26a0 <memcpy+144>
0x269c <memcpy+140>:    movel d2,d1
0x269e <memcpy+142>:    bra 0x26a4 <memcpy+148>
0x26a0 <memcpy+144>:    movel #3,d4
0x26a2 <memcpy+146>:    andl d4,d1
0x26a4 <memcpy+148>:    subl d1,d2
0x26a6 <memcpy+150>:    moveb a0@-,a1@-
0x26a8 <memcpy+152>:    subl #1,d1
0x26aa <memcpy+154>:    bne 0x26a6 <memcpy+150>
0x26ac <memcpy+156>:    movel d2,d1
0x26ae <memcpy+158>:    lsrl #2,d1
0x26b0 <memcpy+160>:    beq 0x26b8 <memcpy+168>
0x26b2 <memcpy+162>:    movel a0@-,a1@-
0x26b4 <memcpy+164>:    subl #1,d1
0x26b6 <memcpy+166>:    bne 0x26b2 <memcpy+162>
0x26b8 <memcpy+168>:    movel #3,d1
0x26ba <memcpy+170>:    andl d2,d1
0x26bc <memcpy+172>:    beq 0x26c4 <memcpy+180>
0x26be <memcpy+174>:    moveb a0@-,a1@-
0x26c0 <memcpy+176>:    subl #1,d1
0x26c2 <memcpy+178>:    bne 0x26be <memcpy+174>
0x26c4 <memcpy+180>:    movel d3,d0
0x26c6 <memcpy+182>:    moveml fp@(-12),d2-d4
0x26cc <memcpy+188>:    unlk fp
0x26ce <memcpy+190>:    rts
```

The machine code of the above program is:

```
<memcpy>:        0x4e56  0x0000  0x48e7  0x3800  0x262e  0x0008  0x242e  0x0010
<memcpy+16>:     0x2243  0x206e  0x000c  0x6700  0x009c  0xb1c3  0x6700  0x0096
<memcpy+32>:     0x634a  0x2208  0x2001  0x8083  0x7803  0xc084  0x6724  0x2001
<memcpy+48>:     0xb780  0x7803  0xc084  0x6606  0x7803  0xb882  0x6504  0x2202
<memcpy+64>:     0x6008  0x7003  0xc081  0x7204  0x9280  0x9481  0x12d8  0x5381
<memcpy+80>:     0x66fa  0x2202  0xe489  0x6706  0x22d8  0x5381  0x66fa  0x7203
<memcpy+96>:     0xc282  0x6750  0x12d8  0x5381  0x66fa  0x6048  0xd1c2  0xd3c2
<memcpy+112>:    0x2208  0x2009  0x8081  0x7803  0xc084  0x6720  0x2009  0xb380
<memcpy+128>:    0x7803  0xc084  0x6606  0x7804  0xb882  0x6504  0x2202  0x6004
<memcpy+144>:    0x7803  0xc284  0x9481  0x1320  0x5381  0x66fa  0x2202  0xe489
<memcpy+160>:    0x6706  0x2320  0x5381  0x66fa  0x7203  0xc282  0x6706  0x1320
<memcpy+176>:    0x5381  0x66fa  0x2003  0x4cee  0x001c  0xfff4  0x4e5e  0x4e75
```

```
'(78      86      0       0       72      231     56      0
  38      46      0       8       36      46      0       16
  34      67      32      110     0       12      103     0
  0       156     177     195     103     0       0       150
  99      74      34      8       32      1       128     131
  120     3       192     132     103     36      32      1
  183     128     120     3       192     132     102     6
  120     3       184     130     101     4       34      2
  96      8       112     3       192     129     114     4
  146     128     148     129     18      216     83      129
  102     250     34      2       228     137     103     6
  34      216     83      129     102     250     114     3
  194     130     103     80      18      216     83      129
  102     250     96      72      209     194     211     194
  34      8       32      9       128     129     120     3
  192     132     103     32      32      9       179     128
  120     3       192     132     102     6       120     4
  184     130     101     4       34      2       96      4
  120     3       194     132     148     129     19      32
  83      129     102     250     34      2       228     137
  103     6       35      32      83      129     102     250
  114     3       194     130     103     6       19      32
  83      129     102     250     32      3       76      238
  0       28      255     244     78      94      78      117)
|#

; in the logic, the above program is defined by (memcpy-code).
```

DEFINITION:

MEMCPY-CODE

```
=  '(78 86 0 0 72 231 56 0 38 46 0 8 36 46 0 16 34 67 32
     110 0 12 103 0 0 156 177 195 103 0 0 150 99 74 34 8
     32 1 128 131 120 3 192 132 103 36 32 1 183 128 120 3
     192 132 102 6 120 3 184 130 101 4 34 2 96 8 112 3
     192 129 114 4 146 128 148 129 18 216 83 129 102 250
     34 2 228 137 103 6 34 216 83 129 102 250 114 3 194
     130 103 80 18 216 83 129 102 250 96 72 209 194 211
     194 34 8 32 9 128 129 120 3 192 132 103 32 32 9 179
     128 120 3 192 132 102 6 120 4 184 130 101 4 34 2 96
     4 120 3 194 132 148 129 19 32 83 129 102 250 34 2
     228 137 103 6 35 32 83 129 102 250 114 3 194 130 103
     6 19 32 83 129 102 250 32 3 76 238 0 28 255 244 78
     94 78 117)

; the initial state.
```

DEFINITION:
memcpy-statep (s, *str1*, n, *lst1*, *str2*, *lst2*)
= ((mc-status (s) = 'running)
∧ evenp (mc-pc (s))
∧ rom-addrp (mc-pc (s), mc-mem (s), 192)
∧ mcode-addrp (mc-pc (s), mc-mem (s), MEMMOVE-CODE)
∧ ram-addrp (sub (32, 16, read-sp (s)), mc-mem (s), 32)
∧ ram-addrp (*str1*, mc-mem (s), n)
∧ mem-lst (1, *str1*, mc-mem (s), n, *lst1*)
∧ ram-addrp (*str2*, mc-mem (s), n)
∧ mem-lst (1, *str2*, mc-mem (s), n, *lst2*)
∧ disjoint (sub (32, 16, read-sp (s)), 32, *str1*, n)
∧ disjoint (sub (32, 16, read-sp (s)), 32, *str2*, n)
∧ (*str1* = read-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
∧ (*str2* = read-mem (add (32, read-sp (s), 8), mc-mem (s), 4))
∧ (n = uread-mem (add (32, read-sp (s), 12), mc-mem (s), 4))
∧ uint-rangep (nat-to-uint (*str1*) + n, 32)
∧ uint-rangep (nat-to-uint (*str2*) + n, 32))

```
; the time function.
```

DEFINITION:
memcpy-t (*str1*, n, *lst1*, *str2*, *lst2*) = memmove-t (*str1*, n, *lst1*, *str2*, *lst2*)

```
; the behavior.
```

DEFINITION:
memcpy (*str1*, *str2*, n, *lst1*, *lst2*) = memmove (*str1*, *str2*, n, *lst1*, *lst2*)

```
; memcpy and memmove are identical.
```

THEOREM: memcpy-memmove-statep
memcpy-statep (s, *str1*, n, *lst1*, *str2*, *lst2*)
= memmove-statep (s, *str1*, n, *lst1*, *str2*, *lst2*)

```
; the correctness.
```

THEOREM: memcpy-correctness
let *sn* **be** stepn (s, memcpy-t (*str1*, n, *lst1*, *str2*, *lst2*))
in
memcpy-statep (s, *str1*, n, *lst1*, *str2*, *lst2*)
→ ((mc-status (*sn*) = 'running)
∧ (mc-pc (*sn*) = rts-addr (s))
∧ (read-rn (32, 14, mc-rfile (*sn*))
= read-rn (32, 14, mc-rfile (s))))

$\wedge$ (read-rn (32, 15, mc-rfile (*sn*))
= add (32, read-sp (*s*), 4))
$\wedge$ ((d2-7a2-5p (*rn*) $\wedge$ (*oplen* $\leq$ 32))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))))
$\wedge$ ((disjoint (*x*, *k*, sub (32, 16, read-sp (*s*)), 32)
$\wedge$ disjoint (*x*, *k*, *str1*, *n*))
$\rightarrow$ (read-mem (*x*, mc-mem (*sn*), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))
$\wedge$ (read-dn (32, 0, *sn*) = *str1*)
$\wedge$ mem-lst (1,
str1,
mc-mem (*sn*),
n,
memcpy (*str1*, *str2*, *n*, *lst1*, *lst2*))) **endlet**

EVENT: Disable memcpy-t.

```
; some properties of memcpy.
; the same as memmove.
```

Index