The "USR" PROC that follows is a "best effort" to simulate a BASIC USR call from an ACTION program.

Presuming that the user has a routine that has been used successfully as a USR call from BASIC, and presuming that the routine has been placed into an ACTION array or PROC, then this "USR" may be called, so long as the address of the array or PROC, the count of parameters (0 to 6), and all the parameters are passed properly.

Finally, a warning:

If you use characters in a string, as in a BASIC line such as this:

```
JUNK = USR( ADR("hhhpLV?"),32)
```

and wish to use that same string in ACTION, watch out!

First, you should use an initialized string, thus:

```
BYTE ARRAY Call(0)="hhhpLV?"
```

and then, when you call the above USR or FUSR routines, you MUST add 1 to the name of the string, because ACTION stores the length of a string as the first byte of that string! Again, something like this might work:

```
USR( Call+1 , 1, 32 )
```

No guarantees that ALL routines callable from BASIC will work with my USR and FUSR, but I'll bet most will. If possible, though, modify the assembly language for a direct ACTION interface. Or, in many cases, if you know what the code is doing, you can rewrite it completely in ACTION with little loss of speed but a BIG gain in readability.

Good luck.

```assemble
PROC USR=*( CARD address
BYTE count
CARD P1,P2,P3,P4,P5,P6 )
[$85$A0 ; STA $A0  Save both bytes of
[$86$A1 ; STX $A1  ...USR address
```
$84$A2 ; STY $A2$ and cnt of parameters
$A2$00 ; LDX #$00 start at beginning
  ;LOOP (a label in assy lang)
$88$ ; DEY Any more parameters?
$30$0A ; BMI DONE No...so go to USR
$B5$A3 ; LDA $A3,X$ Get a parameter
$48$ ; PHA (low byte first)
$B5$A4 ; LDA $A4,X$ and put it on stack
$48$ ; PHA (high byte also)
$E8$ ; INX To next parameter
$E8$ ; INX (each is 2 bytes)
$D0$F3 ; BNE LOOP ALWAYS branch!
  ;DONE (a label in assy lang)
$A5$A2 ; LDA $A2$ get count of parameters
$48$ ; PHA on stack...as USR needs
$6C$A0$00 ; JMP ($A0)$ and thus to the USR
]

; Exception to the above: If the BASIC
; "USR" routine was truly a function...
; that is, if it actually returned a
; value that was used (as opposed to
; the more common "junk" values returned)
; ...then the ACTION program should call
; this "FUSR" routine which moves the
; return value for proper ACTION function
; access.
;
CARD FUNC FUSR(CARD address
  BYTE count
  CARD P1,P2,P3,P4,P5,P6)

CARD rtnval=$D4
USR( address,count,P1,P2,P3,P4,P5,P6)
RETURN (rtnval)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
/===END OF USR and FUSR===
;
</verbatim>

----+ A Demo of the USR Proc

<verbatim>
MODULE

;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
A demo of all the above!
;
We use a BASIC USR call that takes
two numbers and performs a bitwise
AND operation on them, returning the
result.

;
Obviously, this is better done with native ACTION code, but this is for illustration purposes only.

BYTE ARRAY BitAND(0)=
[
$68 ; PLA cnt of parameters
$C9$02 ; CMP #2 only accept proper #
$D0$FE ; BNE * LOOP FOREVER IF WRONG
$68 ; PLA (high byte first... $85$D5 ; STA $D5 where result goes, anyway $68 ; PLA (low byte, ditto) $85$D4 ; STA $D4 end of first argument $68 ; PLA high byte, second arg $25$D5 ; AND $D5 second ANDed with 1st $85$D5 ; STA $D5 is now result! $68 ; PLA similarly for low byte $25$D4 ; AND $D4 $85$D4 ; STA $D4 $60 ; RTS th-th-that's all, folks
]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; A simple test routine

PROC TestUSR()

    CARD Temp1,Temp2

    Temp1 = $614E & $93EB
    PrintF( "Native ACTION: %H%E%E", Temp1 )

    Temp2=FUSR( BitAND,2,$614E,$93EB )
    PrintF( " Using FUSR: %H%E%E", Temp2 )

RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Finally, a warning:

; If you use characters in a string,
; as in a BASIC line such as this:
; JUNK = USR( ADR("hhhpLV"),32)
; and wish to use that same string in
; ACTION, watch out!
;
; First, you should use an initialized
; string, thus:
;
    BYTE ARRAY Call(0)="hhhpLV"
and then, when you call the above
USR or FUSR routines, you MUST add
1 to the name of the string, because
ACTION stores the length of a string
as the first byte of that string!
Again, something like this might work:
USR( Call+1 , 1, 32 )

^^

No guarantees that ALL routines
callable from BASIC will work with
my USR and FUSR, but I'll bet most
will. If possible, though, modify
the assembly language for a direct
ACTION interface. Or, in many cases,
if you know what the code is doing,
you can rewrite it completely in
ACTION with little loss of speed but
a BIG gain in readability.

Good luck.