

Procedure Call – Example with Two Local Variables (2)

```

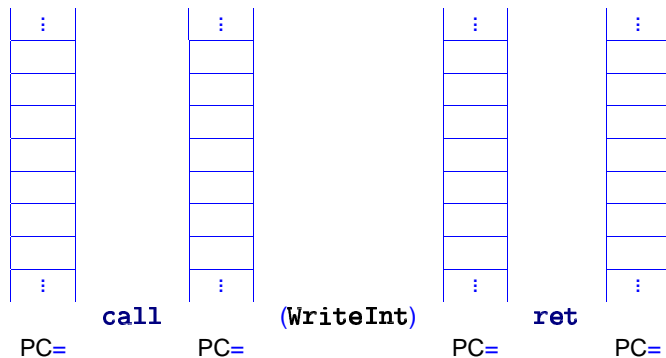
...
; WriteInt(n, 6);
load   n           ; /* Push(n) */ /* AC=Mem[n] */
incsp  #1          ; /* SP=SP+1 */
store  !0          ; /* Mem[SP]=AC */
load   #6          ; /* Push(#6) */ /* AC=6 */
incsp  #1          ; /* SP=SP+1 */
store  !0          ; /* Mem[SP]=AC */
call   WriteInt    ; /* SP=SP+1; Mem[SP]=PC;
                    ; PC=WriteInt */
incsp  #-2         ; /* Pop(2) */ /* SP=SP-2 */

```

Non-void Functions in PeANUt

- we implement local variables via the stack
 - good: economy, privacy, recursion and sharable, re-entrant code and multithread-safe
 - bad: we don't have stack-indexed (or stack-indirect) addressing modes in PeANUt
- inside procedures:
 - we have to increment the SP by number of local variables just after entry (first instruction within procedure)
 - and decrement SP by number of local variables just before (each) **ret**
 - parameters, local variables and return values (RVs) have stack offsets
- non-void function call is similar, but it must first make (empty) slot for the return value, which gets accessed after call before it is popped

Procedure Call – Example with Two Local Variables (3)



Non-void Function – Example with One Return Value (1)

```

RV      = -3      ; int Log10(
x       = -2      ; unsigned int x) {
;
Logx    = 0       ; unsigned int Logx;
NLocs   = 1       ;
;
Log10:  incsp     #NLocs ; /* SP=SP+1 */
        load     !x     ; /* AC=Mem[SP-2] */
        cmp      #0     ; /* compare AC,0 */
        beq     Lendif ;
;
Lendif: ; } /* if */
        load     !Logx  ; return Logx;
        store    !RV    ; /* AC=Mem[SP] */
        incsp   #-NLocs ; /* Mem[SP-3]=AC */
        ret      ; /* SP=SP-1 */
; /* PC=Mem[SP]; SP=SP-1 */
;

```

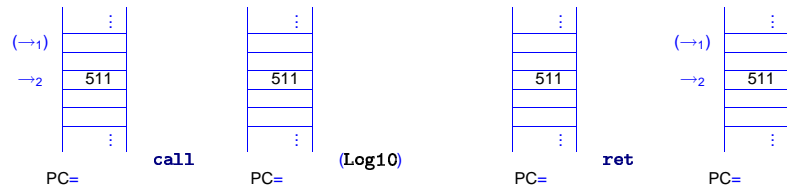
Procedure `Log10()` is available in module `InOut.ass`

Non-void Function – Example with One Return Value (2)

```

log:   block   1       ;   int log;
      ;   log = Log10( 511 );
      incsp   #1       ;   /* SP=SP+1 */ /* make RV slot */
      Push   (#511)   ;   /* AC=511; SP=SP+1; Mem[SP]=AC */
      call   Log10    ;   /* SP=SP+1; Mem[SP]=PC; PC=Log10 */
      Pop    (1)      ;   /* SP=SP-1 */
      load   !0       ;   /* AC=Mem[SP] */ /* store RV */
      store  log      ;   /* Mem[Log]=AC */
      incsp  #-1      ;   /* SP=SP-1 */ /* pop RV slot */

```



Address Parameters – Example (1)

```

      ;   void Sum(
      ;   int a,
      ;   int b,
      ;   int *c) {
Sum:
      load   !c       ;   *c = a + b;
      storexr ;   /* XR=Mem[SP+c] */
      load   !a       ;   /* AC=Mem[SP+a] */
      add    !b       ;   /* AC=AC+Mem[SP+b] */
      store  *0       ;   /* Mem[XR]=AC */
      ret                    } /* Sum() */

      end    main

```

Code is in address-procedure-example.ass

Address Parameters in PeANUt

- in general there are two different types of parameters: value (local copy) and reference (pointers)
- full power of the procedure concept requires the ability to modify data (parameters)
- do via passing address (not value) on stack when calling
- procedures thus reference the *real* memory location indirectly via this pointer

Address Parameters – Example (2)

```

      ;   int
p:     block   1       ;   p,
q:     block   1       ;   q;
main:
      load   #59      ;   p = 59;
      store  p        ;
      ;
      ;   Sum(5, p, &q);
      load   #5       ;   /* Push(5); */
      incsp  #1       ;
      store  !0       ;
      load   p        ;   /* Push(p) */
      incsp  #1       ;
      store  !0       ;
      loada  q        ;   /* Pusha(q) */
      incsp  #1       ;
      store  !0       ;
      call   Sum      ;
      incsp  #-3      ;   /* Pop(3) */

```

