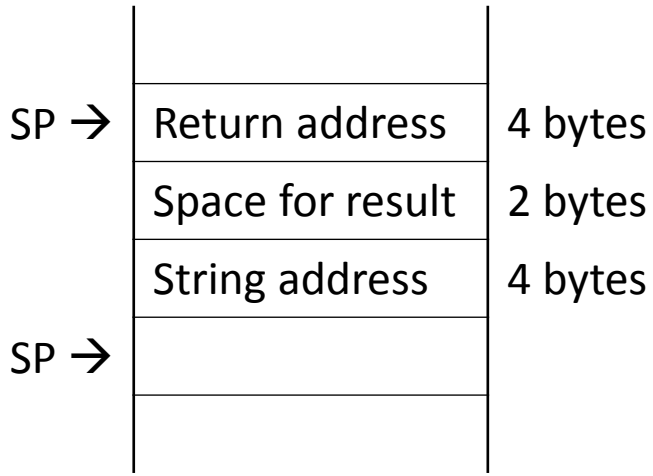
Passing Parameters using Stack

- Calling program pushes parameters on the stack one element at a time before calling subroutine.
- Subroutine Call (jsr, bsr) then pushes the return address on stack.

- Subroutine should therefore provide an additional offset of four to access parameters on stack

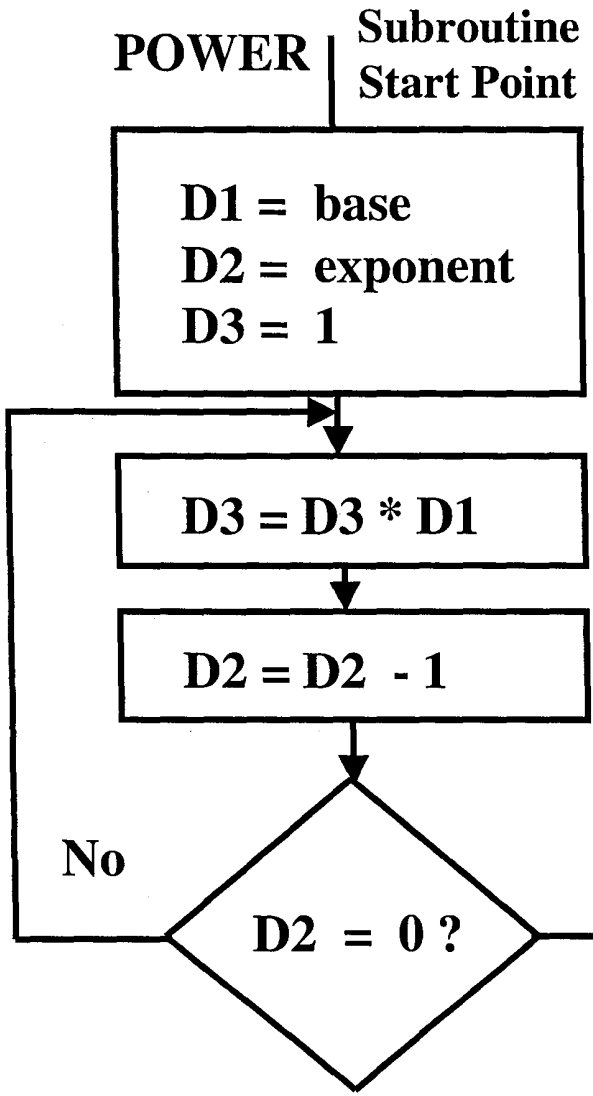
- After returning from subroutine (rts), original parameters are still pushed
- Calling program must increment the stack pointer by the number of bytes the parameters occupy, in order to clean up the stack and bring the Stack Pointer (SP) to its original position.

```
; main program
;
main    equ     *
        …
        …
        lea     string, -(sp)  ; move start address on stack
        clr.w   -(sp)          ; reserve space for result
        jsr     convert        ; call subroutine
        move    (sp)+, d0      ; result is in (sp), save it
        addi    #4, sp         ; clean up the stack
        …
        …
        …                      ; end of code
```

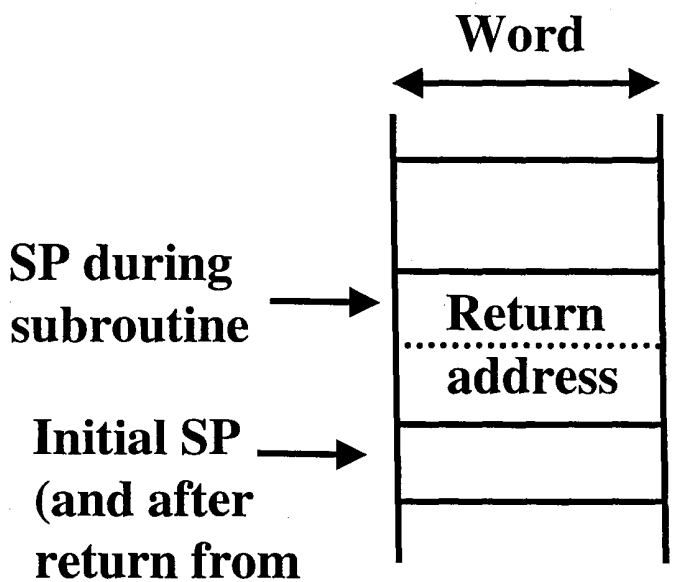



```
; subroutine convert
;
convert equ     *
        movea.l 6(sp), a2  ; first param is at 6(sp)
                           ; second param is at 4(sp)
        ….      ….
        move.w  d1, 4(sp)  ; assume result was in d1
        rts                ; return

;
; data area
;

string  ds.b    20
        end
```

Basic Flow Chart of Power



Effect on The Stack



POWER Subroutine Example (Case 3)

Parameter Passing *by Value*: Using The Stack - Main Program -

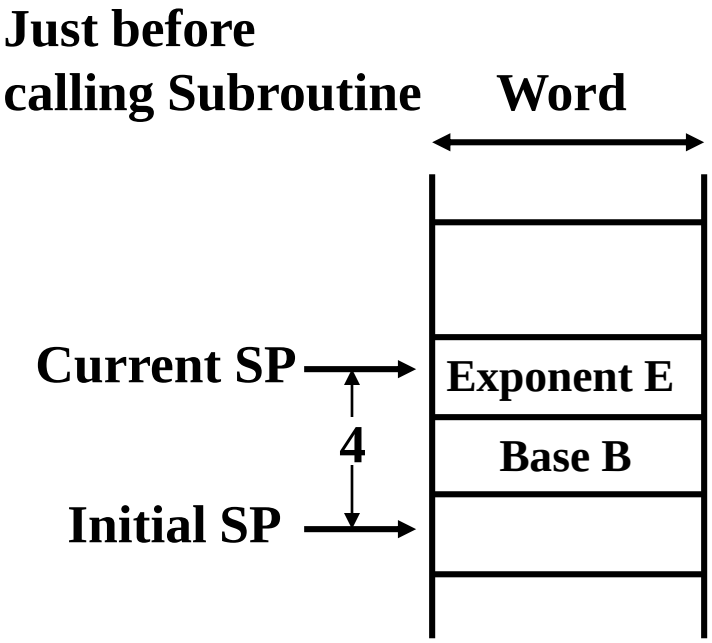
```
MAIN    ORG       $400          Main Program origin
        MOVEA.L   #$07FFE,SP    Initialize  Stack Pointer
        MOVE.B    B,D1          Put base number into D1
        EXT.W     D1            Sign extend base to word length
        MOVE.W    D1,-(SP)      push base  B onto the stack
        CLR.W     D2            Clear D2 before loading exponent
        MOVE.B    E,D2          Put exponent number into D2
        MOVE.W    D2,-(SP)      push exponent E onto the stack
        BSR       POWER         Call subroutine  POWER
        MOVE.L    (SP)+,D3      pop answer from stack resetting SP
        LEA       A,A5           put address of  answer into A5
        MOVE.L    D3,(A5)       save  answer
        MOVE      #228,D7       Done
        TRAP      #14
        ORG       $600
B       DC.B      4             Base number stored here
E       DC.B      2             Exponent number stored here
A       DS.L      1             answer to be stored  here
```

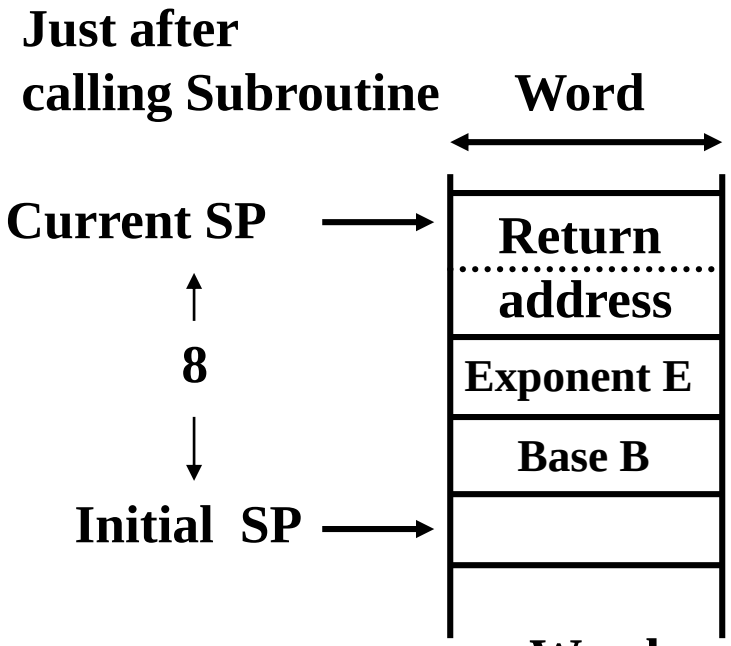
POWER Subroutine Example (Case 3)

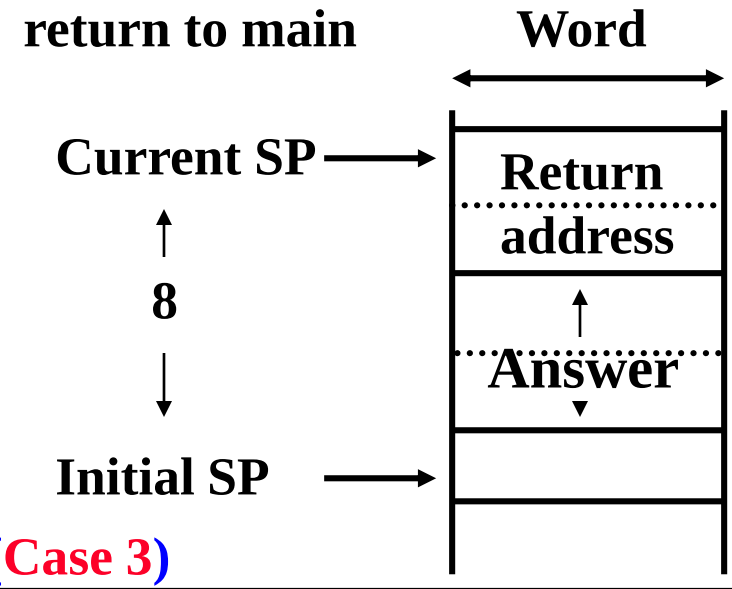
Parameter Passing *by Value*: Using The Stack
Continued - Subroutine -

```
        ORG     $800        Subroutine  POWER origin
POWER   MOVE.W  6(SP),D1    copy base  from stack to D1
        CLR.W   D2          Clear D2 before loading exponent
        MOVE.B  4(SP),D2    copy exponent from to D2
        MOVE.L  #1,D3       initialize result in D3 to 1
LOOP    MULS    D1,D3       multiply result D3 with base D1
        SUB     #1,D2       decrement power in D2 by one
        BNE     LOOP        and repeat as long as power > 0
        MOVE.L  D3,4(SP)    Push result onto the stack
        RTS                 Done, return to calling program
```

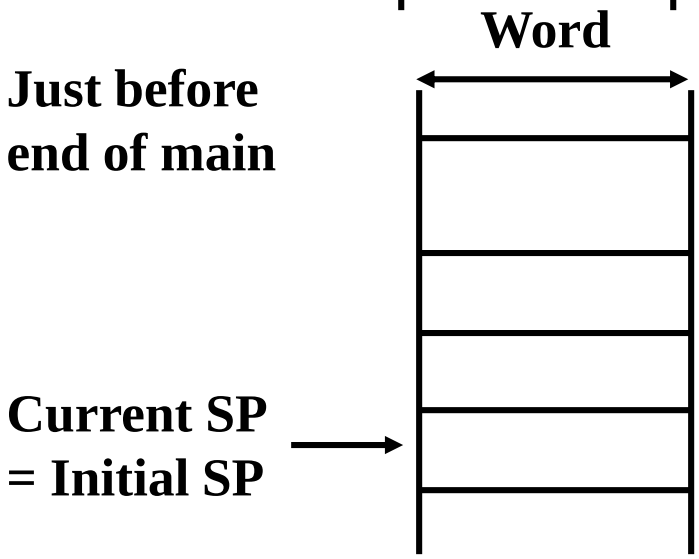
Effect on The Stack







(Case 3)



POWER Subroutine Example (Case 4)

Parameter Passing *by Reference*: Using The Stack

- Main Program -

```
MAIN    ORG       $400          Main Program origin
        MOVEA.L   #$07FFE,SP     Initialize  Stack Pointer
        PEA       B              Push address of Base  onto the stack
        PEA       E              Push address of Exponent  onto the stack
        PEA       A              Push address of Answer  onto the stack
        BSR       POWER          Call subroutine  POWER
        LEA       12(SP),SP      Stack clean-up:  stack pointer reset
        MOVE      #228,D7        Done
        TRAP      #14
        ORG       $600
B       DC.B      4              Base number stored here
E       DC.B      2              Exponent number stored here
A       DS.L      1              answer to be stored  here
```

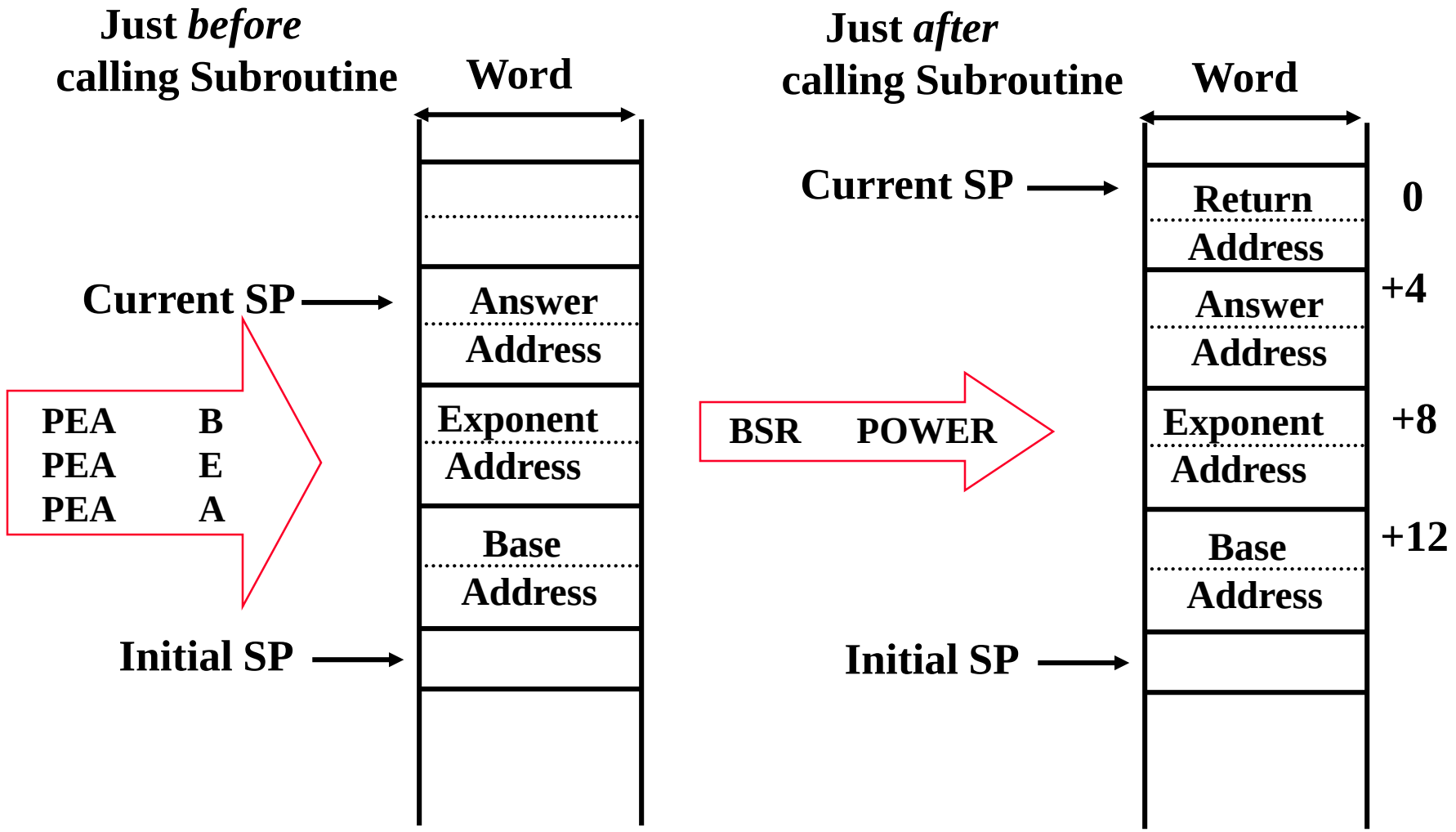
POWER Subroutine Example (Case 4)

Parameter Passing *by Reference*: Using The Stack

Continued - Subroutine -

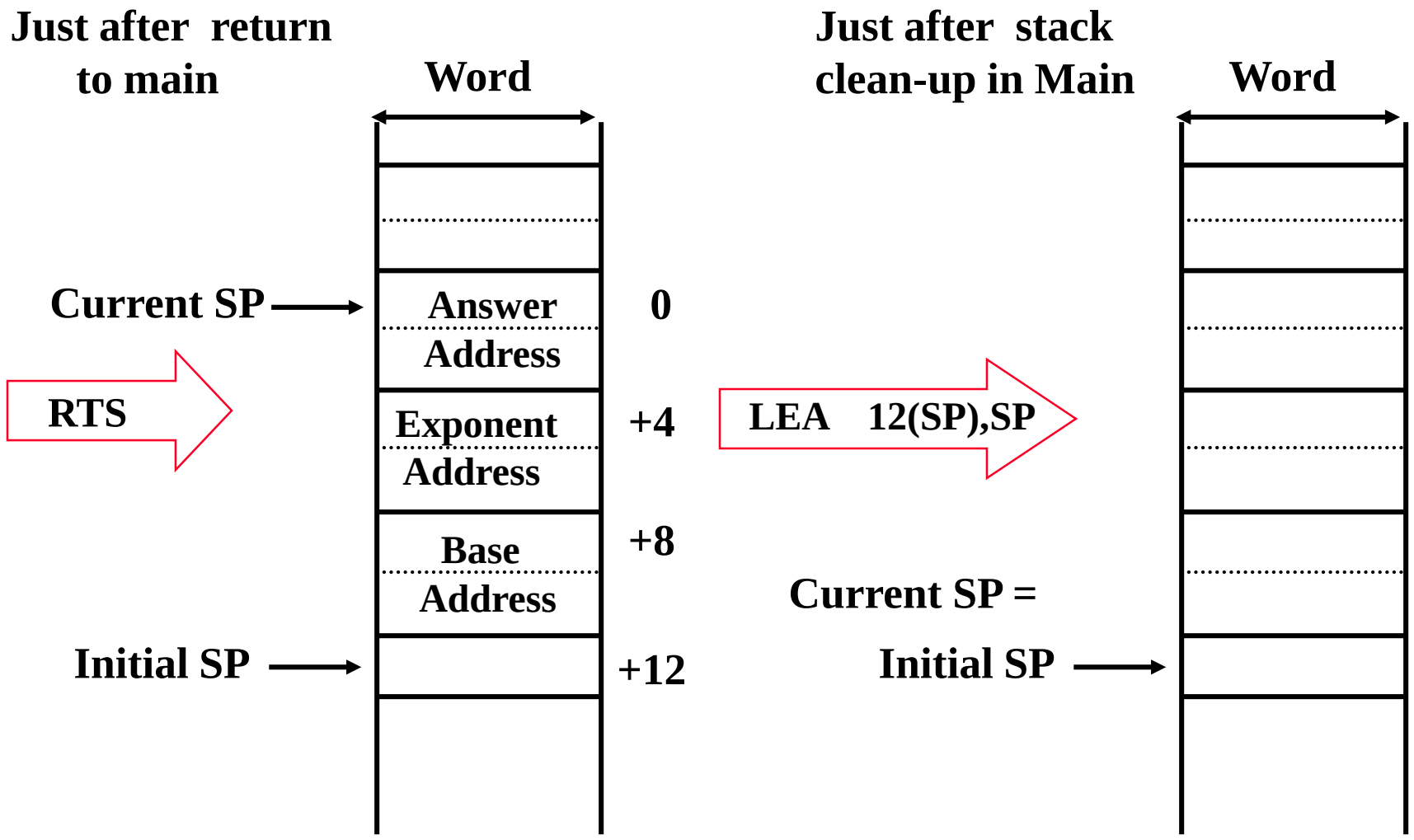
```
        ORG       $800          Subroutine  POWER origin
POWER   MOVEA.L   12(SP),A1     load Base address  in A1
        MOVEA.L   8(SP),A2      load Exponent address  in A2
        MOVEA.L   4(SP),A3      load Answer address address in A3
        MOVE.B    (A1),D1       Put base number into D1
        EXT.W     D1            Sign extend base to word length
        CLR.W     D2            Clear D2 before loading exponent
        MOVE.B    (A2),D2       copy exponent from to D2
        MOVE.L    #1,D3         initialize result in D3 to 1
LOOP    MULS      D1,D3         multiply result D3 with base D1
        SUB       #1,D2         decrement power in D2 by one
        BNE       LOOP          and repeat as long as power  > 0
        MOVE.L    D3,(A3)       Save result in memory
        RTS                     Done, return to calling program
```

Effect on The Stack



(Case 4)

Effect on The Stack

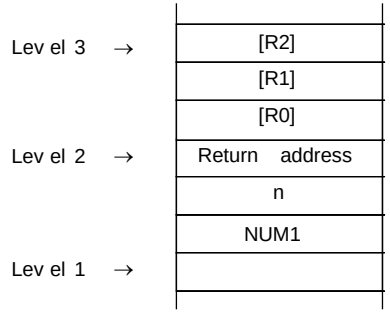


(Case 4)

Assume top of stack is at level 1 below.

```
            Move          #NUM1, -(SP)     Push parameters onto stack.
            Move          N, -(SP)
            Call          LIST ADD         Call subroutine
                                             (top of stack at level 2).
            Move          4(SP),SUM        Save result.
            Add           #8,SP            Restore top of stack
                                             (top of stack at level 1).
            .
            .
            .
LIST ADD    MoveMultiple  R0 - R2, -(SP)   Save registers
                                             (top of stack at level 3).
            Move          16(SP),R1        Initialize counter to n.
            Move          20(SP),R2        Initialize pointer to the list.
            Clear         R0               Initialize sum to 0.
LOOP        Add           (R2)+,R0         Add entry from list.
            Decrement     R1
            Branch > 0    LOOP
            Move          R0,20(SP)        Put result on the stack.
            MoveMultiple  (SP)+,R0 - R2    Restore registers.
            Return                         Return to calling program.
```

(a) Calling program and subroutine



(b) Top of stack at various times

Figure 2.26. Program of Figure 2.16 written as a subroutine; parameters passed on the stack.

Saving/Restoring

- Storage is in the order from a7 to a0, then d7 to d0
- Restore is in the opposite order, first d0 to d7, then a0 to a7

The MOVE Multiple: MOVEM Instruction

- This instruction saves or restores multiple registers.
- Useful in subroutines to save the values of registers not used to pass parameters. MOVEM has two forms:

MOVEM register_list,<ea>

MOVEM <ea>,register_list

- No effect on CCR.

Example: Saving/restoring registers to from memory

```
SUBR1   MOVEM   D0-D7/A0-A6,SAVEBLOCK        SAVE D0-D7/A0-A6
        . . .
        MOVEM   SAVEBLOCK,D0-D7/A0-A6        Restore D0-D7/A0-A6
        RTS
```

Example: Saving/restoring registers using the stack (preferred method).

```
SUBR1   MOVEM   D0-D7/A0-A6,-(SP)        Push D0-D7/A0-A6 onto the stack
        . . .
        MOVEM   (SP)+,D0-D7/A0-A6        Restore D0-D7/A0-A6  from the stack
        RTS
```