Recursive Subroutine Calls Example

The purpose of this example is to examine how all parameters, local variables, return addresses, and frame pointers are stored on the stack when a main program calls a procedure "Process" as well as when the procedure calls itself again in a recursion. We assume the following:

- **The stack pointer initially has the value value $00000F00 just before Process is invoked (before any parameters are pushed onto the stack).**
- **Array "X", "Y", "Z" and "ALPHA" are passed by reference.**
- **Parameter "N" is passed by value (both ways - i.e. into the called procedure and also copied by value back into the calling routine).**
- **A6 is used as the frame pointer (assumed to have initial value $00002000).**
- **Procedure "Process" uses registers D0 - D4 as well as registers A0 - A4.**
- **Array X starts at location $1800, Y starts at $17F8, Z is at $17FC, ALPHA is at $17FD, and N is at $17FE.**

Recursive Subroutine Calls Example

Problem specification (continued):

{main routine}

X: array [0..30] of words

Y: longword

Z, ALPHA, N: byte

Process(var: X, var: Y, var: Z, var: ALPHA, N)

- **We are to show all the M68000 assembly language instructions necessary to pass these parameters as well as to copy the return value N into its regular storage location (off the stack) (at $17FE).**

Recursive Subroutine Calls Example

Problem specification (continued):

Procedure Process (A, B, C, D, E)

A: array [0..?] of words {passed by reference}

B: longword {passed by reference}

C, D: byte {passed by reference}

E: byte {passed both ways by value}

local variables -

T: longword

U: word

V: byte

{ some place within the first invocation of "Process" it calls itself as follows:}

Process(var: A, var: T, var: C, var: V, E) {Note that some input parameters are passed through to the next iteration.}

Recursive Subroutine Calls Example Solution

The main program is assumed to allocate the original storage for:

```
            ORG  $17F8
Y           DS.L   1        This will resolve to address $000017F8
Z           DS.B   1        This will resolve to address $000017FC
ALPHA       DS.B   1        This will resolve to address $000017FD
N           DS.B   1        This will resolve to address $000017FE
*
            ORG  $1800
X           DS.W  31        an array of longwords 0..30
```

Recursive Subroutine Calls Example

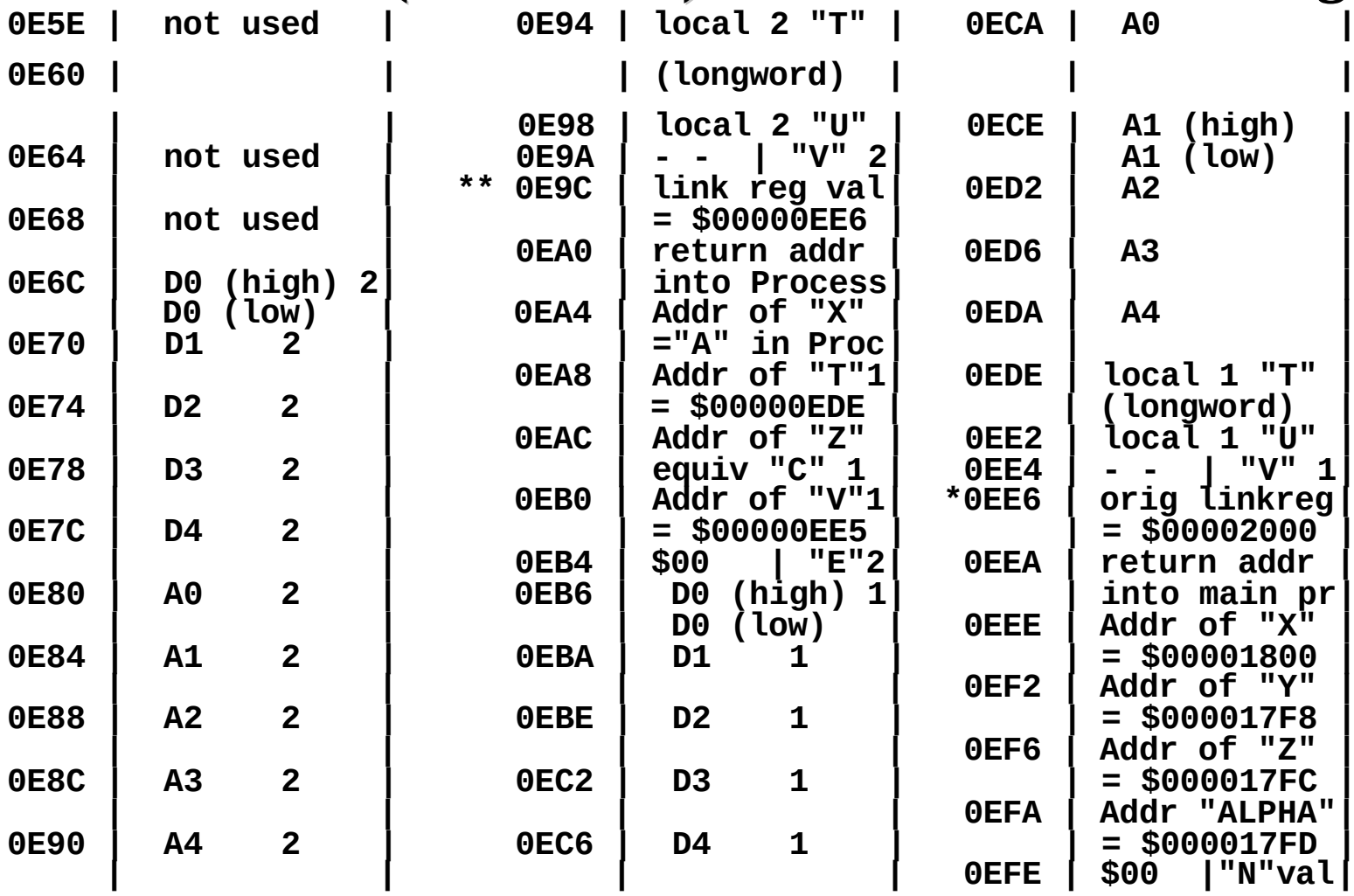
Solution (Continued)

```
        ORG  $1000              (assumed where main program starts - not critical)
*
*   In main program the procedure (subroutine) is called in HLL:
*
*   Process ( var:X, var:Y, var:Z, var:ALPHA, N) where N is the only one passed by value
*   The assembly language version/translation of this invocation is:
*
        CLR.W  D2               zeroes out an entire word for pushing on stack
        MOVE.B N,D2             copies value of byte N into lowest byte of D2
        MOVE.W D2,-(A7)         pushes that word containing value of N on stack
        PEA    ALPHA            pushes pointers to other arguments in reverse
        PEA    Z                order
        PEA    Y
        PEA    X
        JSR    Process          actually call the subroutine here
        MOVE.B 17(A7),N         copy returned value back into N
        ADDA.L #18,A7           fix up stack from all parameters pushed for
*                               subroutine call.
```

Recursive Subroutine Calls Example

Solution (Continued)

Stack Utilization Diagram



*** indicates the value of link register A6 during first call of Process**

**** indicates the value of link register A6 during the second call to Process**

Recursive Subroutine Calls Example Solution (Continued) procedure Process

- **The coding of procedure Process would be something like this:**

```
Procedure Process ( var:A, var:B, var:C, var:D, E )

*  where A:  is an array of words [0.. ?]   passed by reference
*       B:   longword passed by reference
*       C, D:   byte passed by reference
*       E:  byte passed by value (in BOTH directions)
*  and local variables:
*       T:  longword
*       U:  word
 *      V:  byte
Aptr        equ         8          displacements for finding pass by reference
Bptr        equ         12         addresses from the frame pointer: A6
Cptr        equ         16
Dptr        equ         20
E           equ         25         this one is a byte which is passed by value
V           equ         -1
U           equ         -4
T           equ         -8
```

Recursive Subroutine Calls Example
Solution (Continued) procedure Process

```
*  The start of the code of Process looks like this:
*
Process     LINK      A6,#-8
            MOVEM.L   D0-D4/A0-A4,-(A7)          save registers as required
*
*  The invocation of Process from within Process:
*
*  Process ( A, T, C, V, E)
*
            CLR.W     D0
            MOVE.B    E(A6),D0          note how we access  "E" - we could have
            MOVE.W    D0,-(A7)          modified  "E" before sending it
            PEA       V(A6)             this is basically how we can use  "V" too
            MOVE.L    Cptr(A6),-(A7)    we push the pointer to  "Z" on stack
            PEA       T(A6),A0          push pointer to local variable  "T" on stack
            MOVE.L    Aptr(A6),-(A7)    push pointer to  "X"  ("A" in Process)
            JSR       Process
            MOVE.B    17(A7),E(A6)       copy return value of  "E" to local copy
            ADDA.L    #18,A7            fix up stack from all parameters pushed
*
```

Recursive Subroutine Calls Example
Solution (Continued) procedure Process

```
*   This is how we'd access some of the variables in the subroutine:
*
            MOVEA.L   Aptr(A6),A0       This is how we'd copy the first array
             MOVE.L    (A0),U(A6)        element of  X ("A" in procedure) into  "U"
*
            MOVEA.L  Bptr(A6),A1        This is how we'd copy input parameter  "B"
            MOVE.W   (A1),T(A6)         into local word  "T"
*
            MOVEA.L  Cptr(A6),A2        This is how we actually reference  "C"
            MOVE.B   (A2),D1
*
            MOVEA.L  Dptr(A6),A3        This is how we could access/change
            CLR.B      (A3)             "D"  in procedure  = "ALPHA" in main
*
*   Before leaving the procedure we'd need to restore registers and destroy stack frame:
*
            MOVEM.L (A7)+,D0-D4/A0-A4
            UNLK    A6
            RTS
```