The following recursive C function:

```c
int silly(int n, int *p)
{
    int val, val2;

    if (n > 0)
        val2 = silly(n << 1, &val);
    else
        val = val2 = 0;

    *p = val + val2 + n;

    return val + val2;
}
```

yields the following assembly code:

```
silly:
pushl %ebp
movl %esp, %ebp
subl $16, %esp
movl %ebx, -4(%ebp)
movl 8(%ebp), %ebx
testl %ebx, %ebx
jle .L2
leal -8(%ebp), %eax
movl %eax, 4(%esp)
leal (%ebx,%ebx), %eax
movl %eax, (%esp)
call silly ************ here
.L4:
movl -8(%ebp), %edx
addl %edx, %eax
movl 12(%ebp), %edx
leal (%ebx,%eax), %ecx
movl %ecx, (%edx)
movl -4(%ebp), %ebx
movl %ebx, %esp
popl %ebp
ret
.p2align 4,,7
.L2:
movl $0, -8(%ebp)
xorl %eax, %eax
jmp .L4
```
Given the call `silly(2, yp)`, draw the state of the registers and the stack immediately preceding the recursive call to `silly`. You may assume that `yp` points to dynamically allocated space large enough to hold an integer.

Hints:

- Identify the location (stack or register) of each variable used in `silly`.
- Mark any space that is unused as “unused.”
- Use identifying names (such as “old value of ebx”) anywhere you do not know the actual value.