HW1 Solutions

1.

`a = 1, b = 1, c = 2, d = 1`

(Note that `op1` modifies only the *local* copies of `a`, `b`.)

2.

COMMENT1: `a[0] = 0, a[1] = 10, a[2] = 100, a[3] = UNKOWN, a[4] = UNKNOWN` COMMENT2: `a[0] = 0, a[1] = 10, a[2] = 100, a[3] = 2, a[4] = 100`

3.1

```
|  return address from main   |
|  address for right after L7 |
|     rbp for main's frame    | <- %rbp
|             2 (n)           |
|             0 (e)           |
|             0 (f)           | <- %rsp
```

Notes:

- `n` is optional. Some compilers and optimization levels (for example, gcc in unoptimized mode (`-O0`)) will spill it to the stack, because they will spill all called arguments to the stack, in case the argument registers need to reused (this is the case in handout01). Some compilers, even in unoptimized mode, will push arguments to the stack only before using argument registers (see the answer to 3.2), in which case `n` would not be on the stack as it is right before L4.
- the position of `e` and `f` can be the opposite of what we show, either is correct and the order depends on the compiler.

3.2

This is for the mul call in L4.

```
|  return address from main   |
|  address for right after L7 |
|     rbp for main's frame    | <- %rbp
|             2 (n)           |
|             0 (e)           |
|             0 (f)           | <- %rsp
```

`%rip` points to the mov instruction generated as a part of line L4. This mov instruction is responsible for writing the return value in `%rax` to the stack location corresponding to `e` above.

Note: under `-O0` (no compiler optimization), n must be on the stack, regardless of the answer to 3.1. That's because the compilation of "comp" is unaware of the implementation of "mul", and thus the assembly for "comp" has to work even if "mul" clobbers `%rsi` (`%rsi` is where register `n` is passed).

At any compiler optimization higher than `-O0` (for example, `-O1` or `-O2`), the compiler will not place `n` on the stack because the implementation of mul does not clobber any registers, and thus `n` can stay in `%rsi`, and then, for L5, move to `%rdi`.

3.3

```
|  return address from main   |
|  address for right after L7 |
|     rbp for main's frame    | <- comp's rbp
|             0 (e)           |
|             0 (f)           |
| address for the mov from L4 |
|   address for comp's rbp    | <- %rbp
|       unknown value (e)     | <- %rsp
```

Some students omitted the unknown value (e) at the bottom, having read handout01 to imply that the prologue is `pushq %rbp; movq %rsp, %rbp`. However, the prologue is normally thought to include *three* lines (`subq $0x8, %rsp`). This was not clearly marked on the original handout; we have updated to clarify.

This distinction would not have affected the grading.

3.4

```
|  return address from main   |
|  address for right after L7 |
|     rbp for main's frame    |  <- %rbp
|             0 (e)           |
|             0 (f)           |
| address for the mov from L4 | <- %rsp
```

`%rip` points to a part of the text segment after the `mul` function. We cannot know what line of C code this corresponds to since it depends on the compiler.

4.

1. Prevent user level processes from manipulating hardware
2. Program will have good portability across machines
3. Processes do not have to have redundant code