Topic V0D

Recursive Functions

Readings: (Section 2.8)

Non-Leaf Procedures

Procedures that call other procedures

- For nested calls, callee needs to:
  - Save on the stack:
    - `ra`
    - callee-saved registers
  - Restore values from the stack before returning

Calling Itself

```
int fact(int n){
     if (n < 1)
          return 1;
     else
          return (n * fact(n – 1));
}
```

Step-by-step creation of the assembly code for fact

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n - 1));
}
```

```
save ra;
if (n < 1){
        restore ra;
        return 1;
}
else{
        tA ← n * fact(n - 1);
        restore ra;
        return tA;
}
```

```
save ra;
if (n < 1){
        restore ra;
        a0 ← 1;
        return a0;
}
else{
        tA ← n * fact(n - 1);
        restore ra;
        return tA;
}
```

```
save ra;
if (n < 1){
        restore ra;
        a0 ← 1;
        return a0;
}
else{
        a0 ← n - 1;
        tA ← n * fact(a0);
        restore ra;
        return tA;
}
```

n is also in a0

```
save ra;
if (n < 1){
        restore ra;
        a0 ← 1;
        return a0;
}
else{
        a0 ← n – 1;
        tA ← n * fact(a0);
        restore ra;
        return tA;
}
```

```
save ra;
if (n < 1){
        restore ra;
        a0 ← 1;
        return a0;
}
else{
        a0 ← n – 1;
        a0 ← fact(a0);
        tA ← n * a0;
        restore ra;
        return tA;
}
```

```
save ra, a0;
if (n < 1){
        restore ra, a0;
        a0 ← 1;
        return a0;
}
else{
        a0 ← n – 1;
        a0 ← fact(a0);
        restore ra, a0;
        tA ← n * a0;
        return tA;
}
```

n is used here

This a0 is n

How to solve this?

```
save ra, a0;
if (n < 1){
        restore ra, a0;
        a0 ← 1;
        return a0;
}
else{
        a0 ← n - 1;
        a0 ← fact(a0);
        restore ra, a0;
        tA ← n * a0;
        return tA;
}
```

```
save ra, a0;
if (a0 < 1){
        restore ra, a0;
        a0 ← 1;
        return a0;
}
else{
        a0 ← a0 - 1;
        a0 ← fact(a0);
        restore ra, a0;
        tA ← a0 * a0;
        return tA;
}
```

n-1 n

```
save ra, a0;
if (a0 < 1){
        restore ra, a0;
        a0 ← 1;
        return a0;
}
else{
        a0 ← a0 - 1;
        a0 ← fact(a0);
        t0 ← a0;
        restore ra, a0;
        a0 ← a0 * t0;
        return a0;
}
```

This a0 is fact(n-1)

What is wrong with this?

This multiplication is n*n

It should be n*fact(n-1)

This is n

This is fact(n-1)

This a0 is n

```
save ra, a0;
if (a0 < 1){
        restore ra, a0;
        a0 ← 1;
        return a0;
}
else{
        a0 ← a0 – 1;
        a0 ← fact(a0);
        t0 ← a0;
        restore ra, a0;
        a0 ← a0 * t0;
        return a0;
}
```

```
sp ← sp – 8;
M[sp + 4] ← ra;
M[sp] ← a0;
if (a0 < 1){
        restore ra, a0;
        a0 ← 1;
        return a0;
}
else{
        a0 ← a0 – 1;
        a0 ← fact(a0);
        t0 ← a0;
        a0 ← M[sp];
        ra ← M[sp + 4];
        sp ← sp + 8;
        a0 ← a0 * t0;
        return a0;
}
```

```
sp ← sp – 8;
M[sp + 4] ← ra;
M[sp] ← a0;
if (a0 < 1){
        ra ← M[sp + 4]
        a0 ← M[sp];
        sp ← sp + 8;
        a0 ← 1;
        return a0;
}
else{
        a0 ← a0 – 1;
        a0 ← fact(a0);
        t0 ← a0;
        a0 ← M[sp];
        ra ← M[sp + 4];
        sp ← sp + 8;
        a0 ← a0 * t0;
        return a0;
}
```

Recursive Procedure

```
sp ← sp – 8;
M[sp + 4] ← ra;
M[sp] ← a0;
if (a0 < 1){
        sp ← sp + 8;
        a0 ← 1;
        return a0;
}
else{
        a0 ← a0 – 1;
        a0 ← fact(a0);
        t0 ← a0;
        a0 ← M[sp];
        ra ← M[sp + 4];
        sp ← sp + 8;
        a0 ← a0 * t0;
        return a0;
}
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Recursive Procedure

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

```
sp ← sp – 8;
M[sp + 4] ← ra;
M[sp] ← a0;
if (a0 < 1){
        sp ← sp + 8;
        a0 ← 1;
        return a0;
}
else{
        a0 ← a0 – 1;
        a0 ← fact(a0);
        t0 ← a0;
        a0 ← M[sp];
        ra ← M[sp + 4];
        sp ← sp + 8;
        a0 ← a0 * t0;
        return a0;
}
```

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

```
int fact(int n){
    save ra;
    if (
    if (
    sp ← sp – 8;
    M[sp + 4] ← ra;
    M[sp] ← a0;
    if (a0 < 1){
        sp ← sp + 8;
        a0 ← 1;
        return a0;
    }
    else{
        a0 ← a0 – 1;
        a0 ← fact(a0);
        t0 ← a0;
        a0 ← M[sp];
        ra ← M[sp + 4];
        sp ← sp + 8;
        a0 ← a0 * t0;
        return a0;
    }
    }
    else
    }
    }
    – 1));
}
```

```
fact:
    addi    sp, sp, -8
    sw      ra, 4(sp)
    sw      a0, 0(sp)
    li      t1, 1
    bge     a0, t1, L1
    addi    a0, zero, 1
    addi    sp, sp, 8
    jalr    zero, ra, 0
L1: addi    a0, a0, -1
    jal     ra, fact
    mv      t0, a0
    lw      a0, 0(sp)
    lw      ra, 4(sp)
    addi    sp, sp, 8
    mul     a0, a0, t0
    jalr    zero, ra, 0
```

Recap

The simulation will not track t1 because it is only stores the value 1 for these instructions:

```
li      t1, 1        # t1 <- 1
bge     a0, t1, L1   # if n ≥ 1, go to L1
```

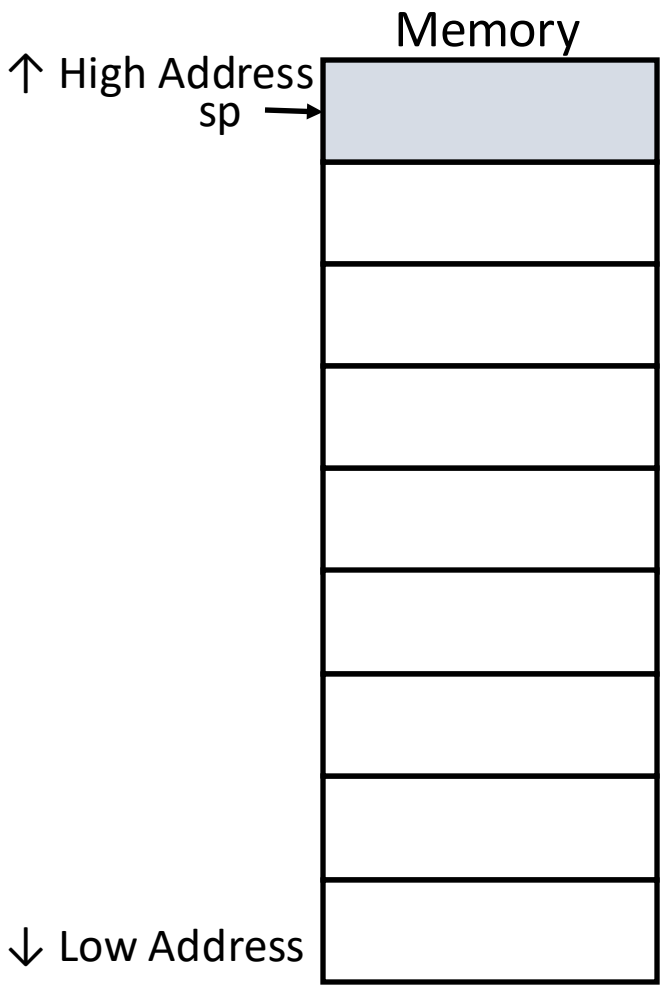
$n \geq 1 \equiv n > 0$ if n is an integer

The two instructions above could be replaced with:

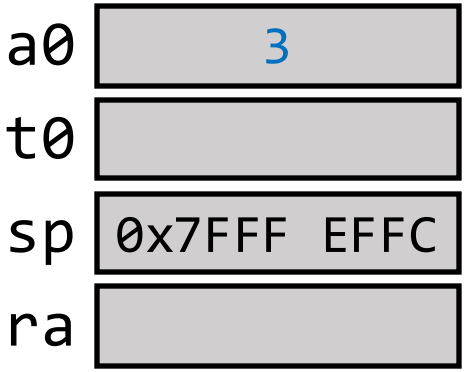
```
blt     zero, a0, L1   # if n > 0, go to L1
```


```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

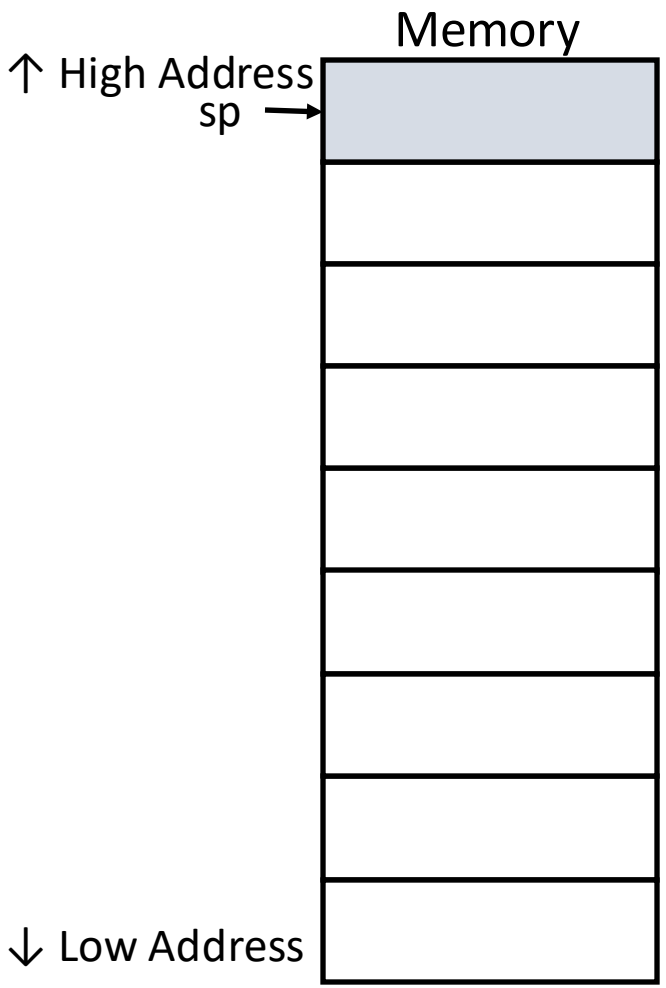



```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```




```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

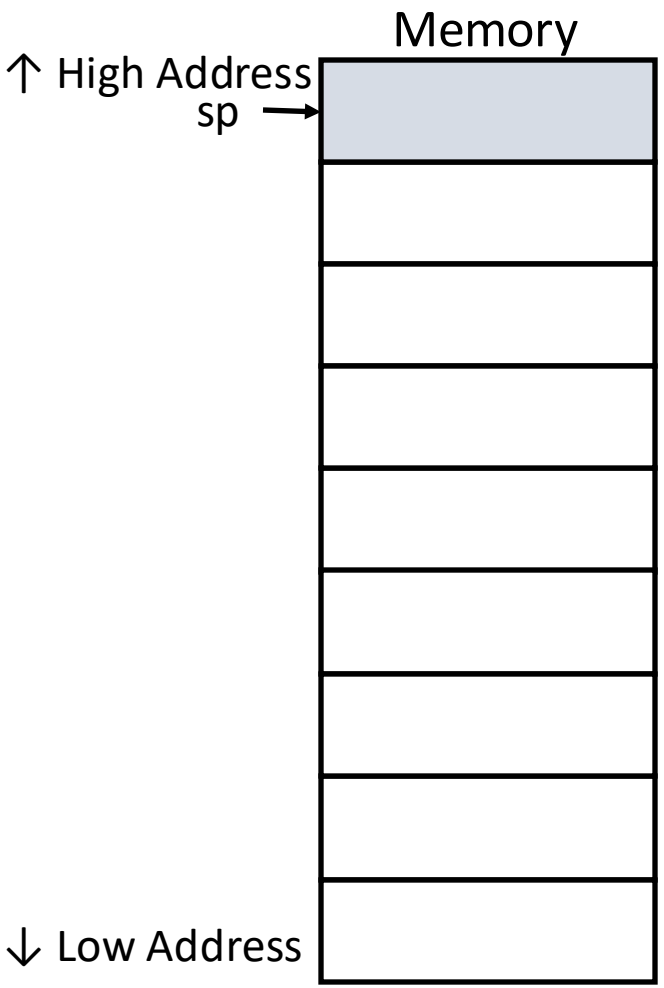



```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	3
t0	
sp	0x7FFF EFFC
ra	0x0040 0008

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

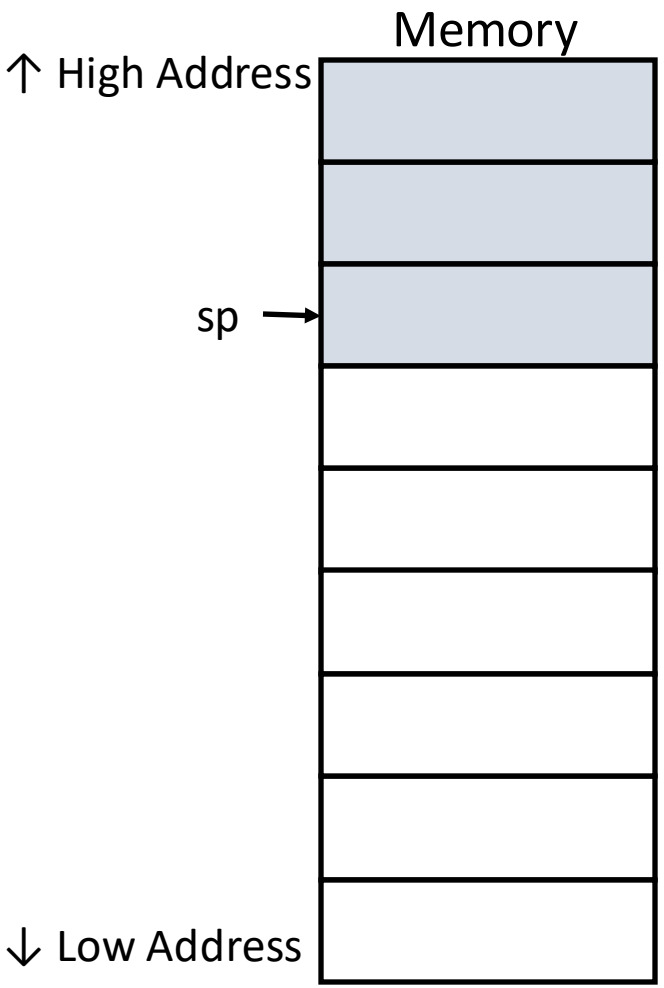



```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	3
t0	
sp	0x7FFF EFF4
ra	0x0040 0008

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```




```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	3
t0	
sp	0x7FFF EFF4
ra	0x0040 0008

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008
3

sp → 3

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	3
t0	
sp	0x7FFF EFF4
ra	0x0040 0008

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008
sp → 3

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	3
t0	
sp	0x7FFF EFF4
ra	0x0040 0008

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008
3

sp → 3

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	2
t0	
sp	0x7FFF EFF4
ra	0x0040 0008

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008
3

sp → 3

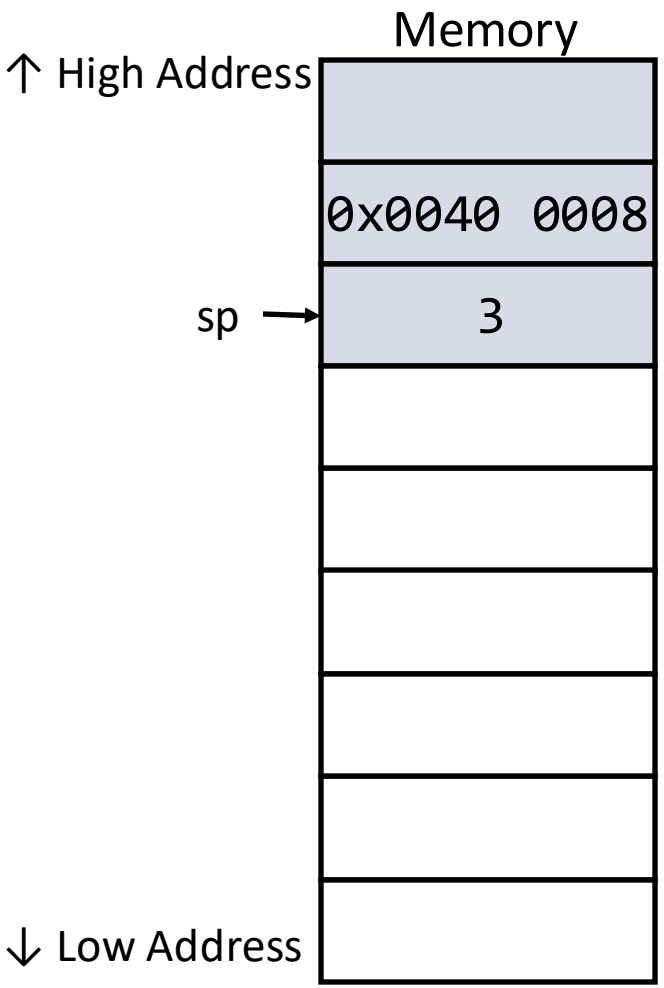
↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	2
t0	
sp	0x7FFF EFF4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```




```
fact:
    addi    sp, sp, -8     # make room in stack for 2 more items
    sw      ra, 4(sp)      # save the return address
    sw      a0, 0(sp)      # save the argument n
    li      t1, 1          # t1 <- 1
    bge     a0, t1, L1     # if n ≥ 1, go to L1
    addi    a0, zero, 1    # return 1
    addi    sp, sp, 8      # pop two items from the stack
    jalr    zero, ra, 0    # return to the instruction after jal
L1: addi    a0, a0, -1     # subtract 1 from the argument
    jal     ra, fact       # jump to fact and save position to ra
    mv      t0, a0         # copy the return value; don't overwrite
    lw      a0, 0(sp)      # just returned from jal, restore n
    lw      ra, 4(sp)      # restore the return address
    addi    sp, sp, 8      # pop two items from the stack
    mul     a0, a0, t0     # return n * fact(n – 1)
    jalr    zero, ra, 0    # return to the caller
```

a0	2
t0	
sp	0x7FFF EFEC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008
3
sp →

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	2
t0	
sp	0x7FFF EFEC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	← sp

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	
sp	0x7FFF EFEC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	← sp

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	
sp	0x7FFF EFEC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	← sp

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	
sp	0x7FFF EFE4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	
0x0040 0040	
1	← sp

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	0
t0	
sp	0x7FFF EFE4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	
0x0040 0040	
1	← sp

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	0
t0	
sp	0x7FFF EFE4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	
0x0040 0040	
1	← sp

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	0
t0	
sp	0x7FFF EFDC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	
0x0040 0040	
1	
0x0040 0040	
0	← sp

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	0
t0	
sp	0x7FFF EFDC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008
3
0x0040 0040
2
0x0040 0040
1
0x0040 0040
0

sp →

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	
sp	0x7FFF EFDC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	
0x0040 0040	
1	
0x0040 0040	
0	← sp

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	
sp	0x7FFF EFE4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	
0x0040 0040	
1	← sp
0x0040 0040	
0	

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	1
sp	0x7FFF EFE4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address	
	0x0040 0008
	3
	0x0040 0040
	2
	0x0040 0040
sp →	1
	0x0040 0040
↓ Low Address	0

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	1
sp	0x7FFF EFE4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address	
	0x0040 0008
	3
	0x0040 0040
	2
	0x0040 0040
sp →	1
	0x0040 0040
↓ Low Address	0

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	1
sp	0x7FFF EFE4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

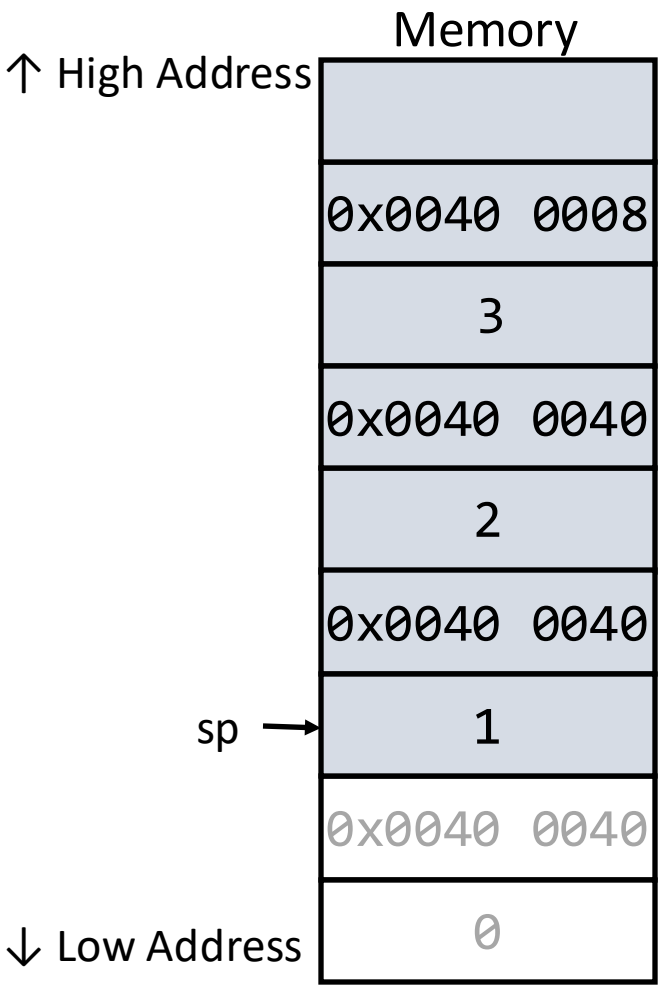
↑ High Address	
	0x0040 0008
	3
	0x0040 0040
	2
	0x0040 0040
sp →	1
	0x0040 0040
↓ Low Address	0

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	1
sp	0x7FFF EFE4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```




```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	1
sp	0x7FFF EFEC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	← sp
0x0040 0040	
1	
0x0040 0040	
0	

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	1
sp	0x7FFF EFEC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	← sp
0x0040 0040	
1	
0x0040 0040	
0	

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	1
sp	0x7FFF EFEC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	← sp
0x0040 0040	
1	
0x0040 0040	
0	

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	1
t0	1
sp	0x7FFF EFEC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	← sp
0x0040 0040	
1	
0x0040 0040	
0	

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	2
t0	1
sp	0x7FFF EFEC
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

0x0040 0008	
3	
0x0040 0040	
2	← sp
0x0040 0040	
1	
0x0040 0040	
0	

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	2
t0	1
sp	0x7FFF EFF4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

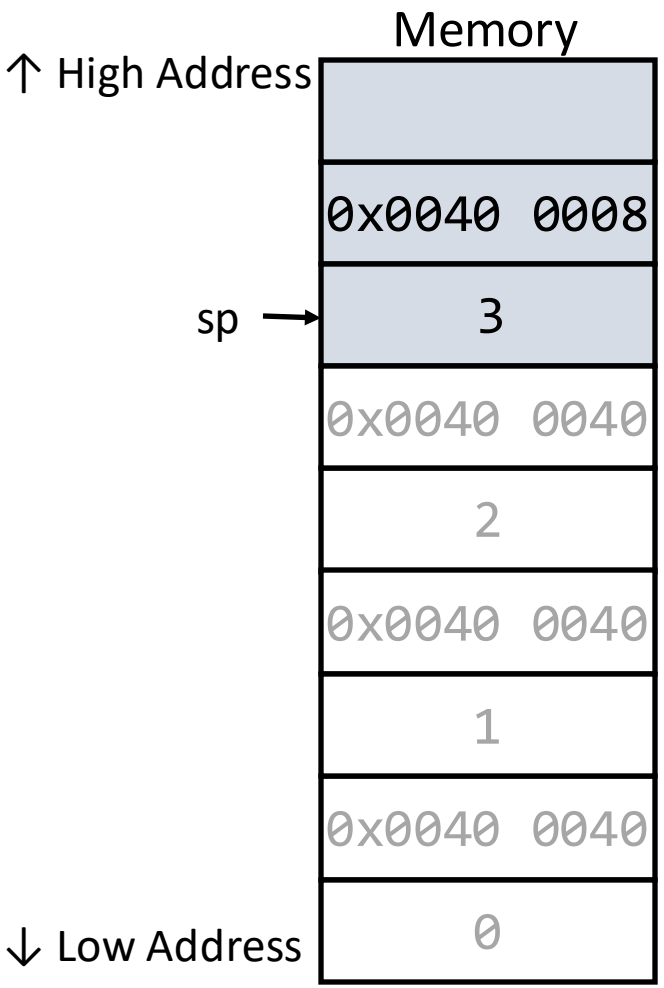
↑ High Address	
	0x0040 0008
sp →	3
	0x0040 0040
	2
	0x0040 0040
	1
	0x0040 0040
↓ Low Address	0

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	2
t0	1
sp	0x7FFF EFF4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```




```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	2
t0	1
sp	0x7FFF EFF4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

	0x0040 0008
sp →	3
	0x0040 0040
	2
	0x0040 0040
	1
	0x0040 0040
	0

↓ Low Address

```
fact:
     addi    sp, sp, -8      # make room in stack for 2 more items
     sw      ra, 4(sp)       # save the return address
     sw      a0, 0(sp)       # save the argument n
     li      t1, 1           # t1 <- 1
     bge     a0, t1, L1      # if n ≥ 1, go to L1
     addi    a0, zero, 1     # return 1
     addi    sp, sp, 8       # pop two items from the stack
     jalr    zero, ra, 0     # return to the instruction after jal
L1:  addi    a0, a0, -1      # subtract 1 from the argument
     jal     ra, fact        # jump to fact and save position to ra
     mv      t0, a0          # copy the return value; don't overwrite
     lw      a0, 0(sp)       # just returned from jal, restore n
     lw      ra, 4(sp)       # restore the return address
     addi    sp, sp, 8       # pop two items from the stack
     mul     a0, a0, t0      # return n * fact(n – 1)
     jalr    zero, ra, 0     # return to the caller
```

a0	2
t0	2
sp	0x7FFF EFF4
ra	0x0040 0040

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

	0x0040 0008
sp →	3
	0x0040 0040
	2
	0x0040 0040
	1
	0x0040 0040
	0

↓ Low Address

```
fact:
     addi    sp, sp, -8      # make room in stack for 2 more items
     sw      ra, 4(sp)       # save the return address
     sw      a0, 0(sp)       # save the argument n
     li      t1, 1           # t1 <- 1
     bge     a0, t1, L1      # if n ≥ 1, go to L1
     addi    a0, zero, 1     # return 1
     addi    sp, sp, 8       # pop two items from the stack
     jalr    zero, ra, 0     # return to the instruction after jal
L1:  addi    a0, a0, -1      # subtract 1 from the argument
     jal     ra, fact        # jump to fact and save position to ra
     mv      t0, a0          # copy the return value; don't overwrite
     lw      a0, 0(sp)       # just returned from jal, restore n
     lw      ra, 4(sp)       # restore the return address
     addi    sp, sp, 8       # pop two items from the stack
     mul     a0, a0, t0      # return n * fact(n – 1)
     jalr    zero, ra, 0     # return to the caller
```

a0	3
t0	2
sp	0x7FFF EFF4
ra	0x0040 0008

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address	
	0x0040 0008
sp →	3
	0x0040 0040
	2
	0x0040 0040
	1
	0x0040 0040
↓ Low Address	0

```
fact:
     addi    sp, sp, -8      # make room in stack for 2 more items
     sw      ra, 4(sp)       # save the return address
     sw      a0, 0(sp)       # save the argument n
     li      t1, 1           # t1 <- 1
     bge     a0, t1, L1      # if n ≥ 1, go to L1
     addi    a0, zero, 1     # return 1
     addi    sp, sp, 8       # pop two items from the stack
     jalr    zero, ra, 0     # return to the instruction after jal
L1:  addi    a0, a0, -1      # subtract 1 from the argument
     jal     ra, fact        # jump to fact and save position to ra
     mv      t0, a0          # copy the return value; don't overwrite
     lw      a0, 0(sp)       # just returned from jal, restore n
     lw      ra, 4(sp)       # restore the return address
     addi    sp, sp, 8       # pop two items from the stack
     mul     a0, a0, t0      # return n * fact(n – 1)
     jalr    zero, ra, 0     # return to the caller
```

a0	3
t0	2
sp	0x7FFF EFFC
ra	0x0040 0008

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

sp →

0x0040 0008
3
0x0040 0040
2
0x0040 0040
1
0x0040 0040
0

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	6
t0	2
sp	0x7FFF EFFC
ra	0x0040 0008

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

sp →

0x0040 0008
3
0x0040 0040
2
0x0040 0040
1
0x0040 0040
0

↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

a0	6
t0	2
sp	0x7FFF EFFC
ra	0x0040 0008

```
0x0040 0000    addi    a0, zero, 3
0x0040 0004    jal     ra, fact
0x0040 0008    …
```

```
int fact(int n){
        if (n < 1)
                return 1;
        else
                return (n * fact(n – 1));
}
```

Memory

↑ High Address

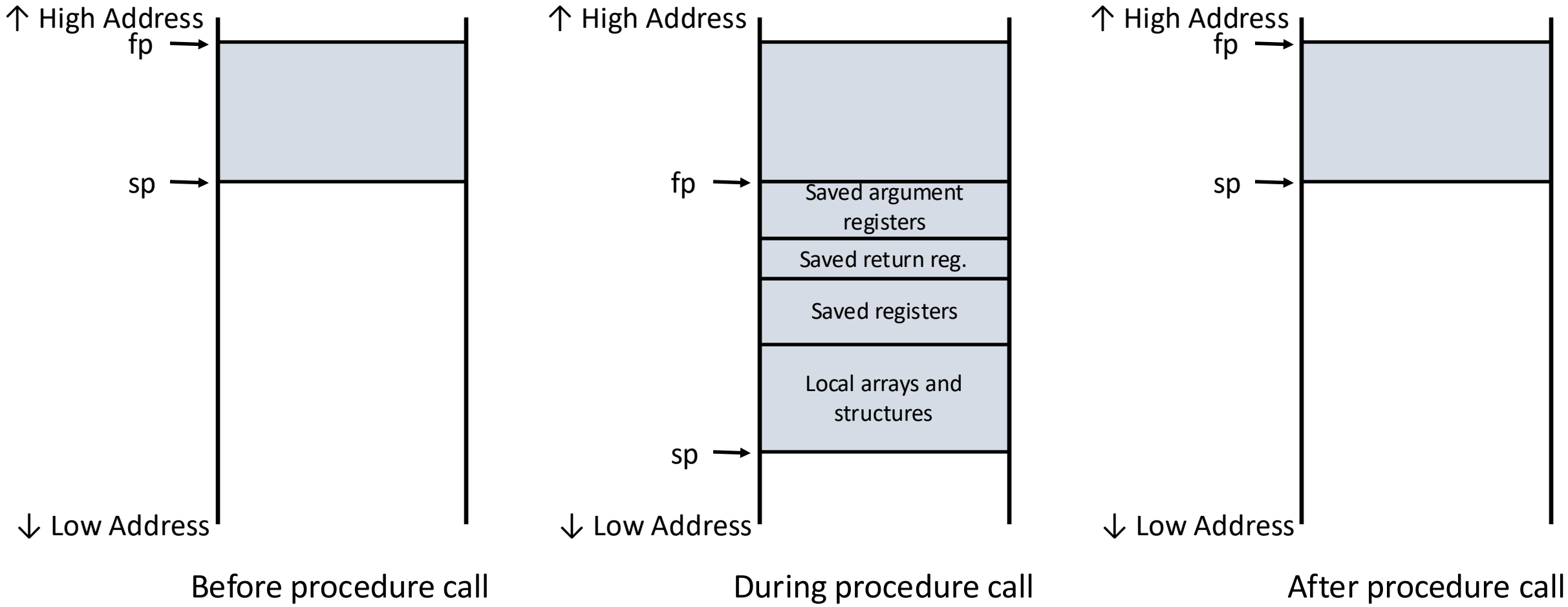
sp →

0x0040 0008
3
0x0040 0040
2
0x0040 0040
1
0x0040 0040
0

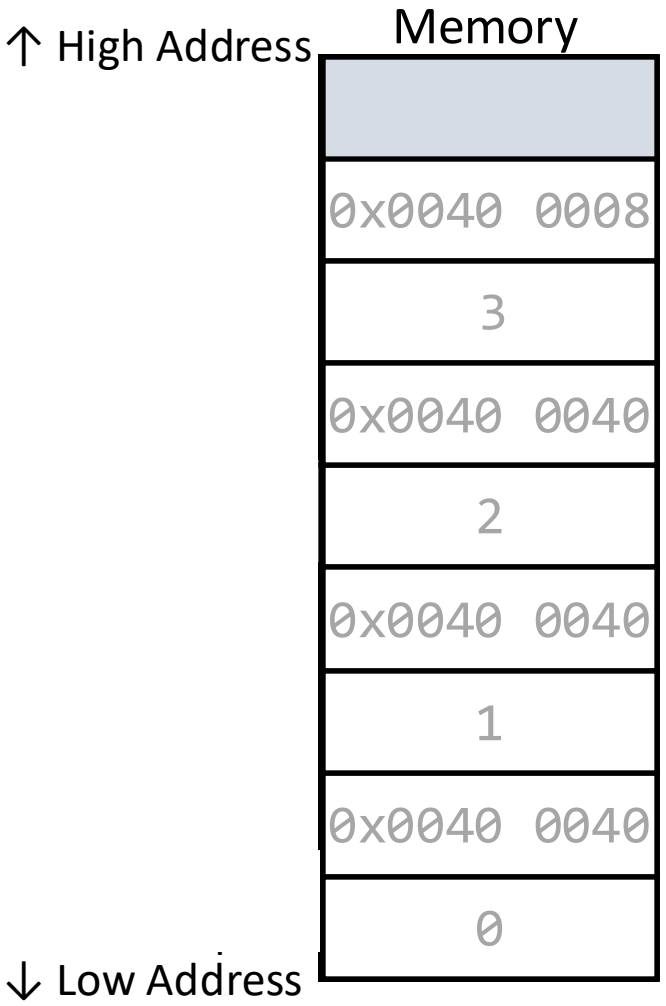
↓ Low Address

```
fact:
    addi    sp, sp, -8      # make room in stack for 2 more items
    sw      ra, 4(sp)       # save the return address
    sw      a0, 0(sp)       # save the argument n
    li      t1, 1           # t1 <- 1
    bge     a0, t1, L1      # if n ≥ 1, go to L1
    addi    a0, zero, 1     # return 1
    addi    sp, sp, 8       # pop two items from the stack
    jalr    zero, ra, 0     # return to the instruction after jal
L1: addi    a0, a0, -1      # subtract 1 from the argument
    jal     ra, fact        # jump to fact and save position to ra
    mv      t0, a0          # copy the return value; don't overwrite
    lw      a0, 0(sp)       # just returned from jal, restore n
    lw      ra, 4(sp)       # restore the return address
    addi    sp, sp, 8       # pop two items from the stack
    mul     a0, a0, t0      # return n * fact(n – 1)
    jalr    zero, ra, 0     # return to the caller
```

Other Data Stored in the Stack



Recap


↑ High Address
Memory
0x0040 0008
3
0x0040 0040
2
0x0040 0040
1
0x0040 0040
0
↓ Low Address